De cero a producción con Docker

May 2018

¡Hola!

Agenda

Día 1 -
¿Qué es Docker? - Conceptos
“Hola Mundo”
Administración a fondo de containers

Día 2 -
Administración de volúmenes
Todo lo que hay que saber de imágenes
Comandos de troubleshooting

Día 3 -
Orquestación de distintos componentes
Manejo de networking
Monitoreo

Día 4 -
El Mundo de Producción
Lecciones aprendidas
Cómo Meli utiliza Docker

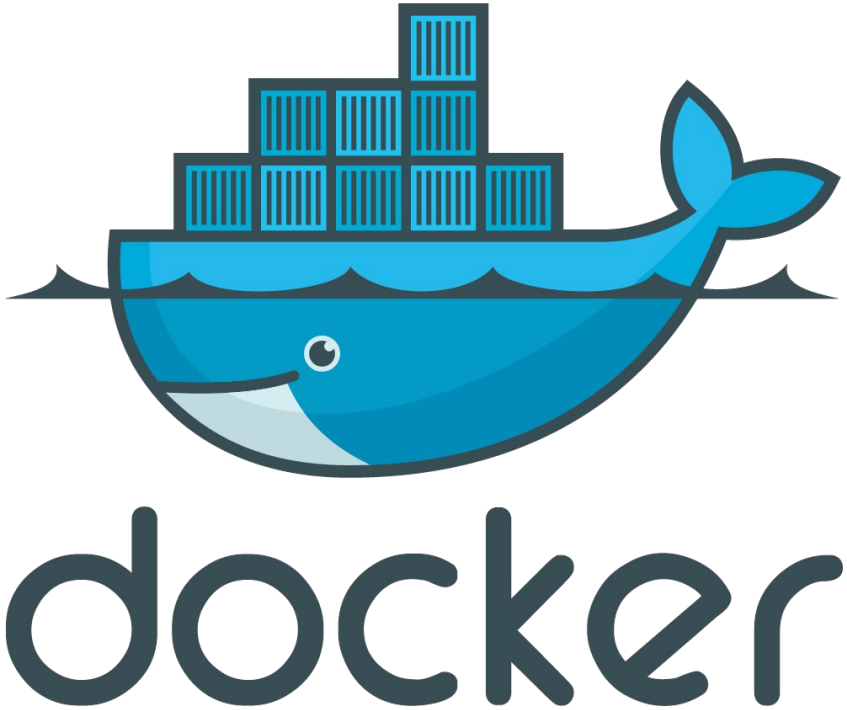
Detalles logísticos

- Recreo de 15/20 minutos para dispersar
- WIFI guest / mercadolibre
- Clona -> git@github.com:mercadolibre/workshop-docker.git
- Cualquier cosa:
  - lucia.brizuela@mercadolibre.com
  - sebastian.schepens@mercadolibre.com

- **Pregunta sobre todo, todo el tiempo**

Día 1

¿Qué es docker?



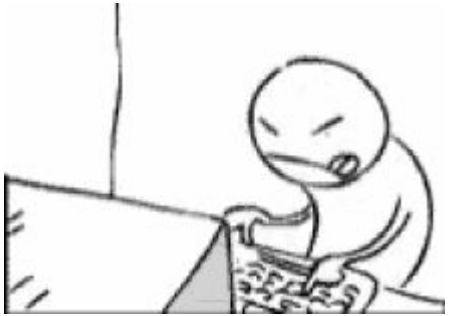

TERITON
PIL
Touax
msc
CMA CGM
EMIRATES
TEREX
SWL 45
97-1617
GO

¿Qué resuelven los containers?

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

ANSIBLE

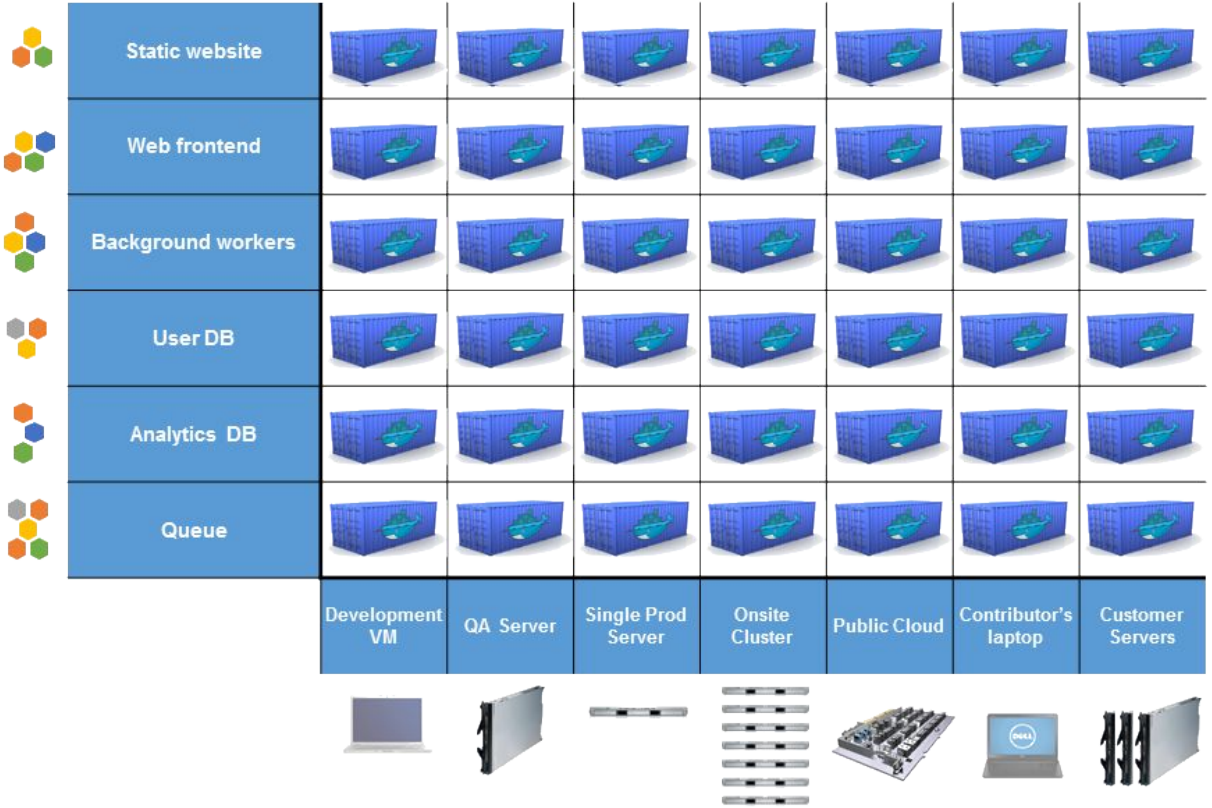
CHEF™



SSH

```
#!/bin/bash
#INPUT_SAMPLE_LIST=$1
cd /Volumes/PhilDrive_EMS/TestDec7/snv_postprocess/

. paths.txt

echo "Debug level set for $DEBUG_LEVEL"
echo "log found in scripts directory"

cp $HIGH_SNP_OUT ./
cp $LOW_SNP_OUT ./
cp $GERM_SNP_OUT ./
#  echo "${SCRIPT_DIR}/run_somatic_mutation_analysis ${1} no_false_snp"
if [ $DEBUG_LEVEL -gt 0 ]
then
echo "INFO: ${SCRIPT_DIR}run_somatic_mutation_analysis.sh $SAMPLE no_false_snp
`basename ${LOW_SNP_OUT}` `basename ${GERM_SNP_OUT}` `basename ${HIGH_SNP_OUT}`
${D_BAM_FILE} ${G_BAM_FILE}\>>${LOG}

fi
${SCRIPT_DIR}run_somatic_mutation_analysis.sh

echo "End of somatic mutation analysis">> $LOG
```

¿Qué resuelven los containers?



Setup

Primer ejercicio

Primer ejercicio

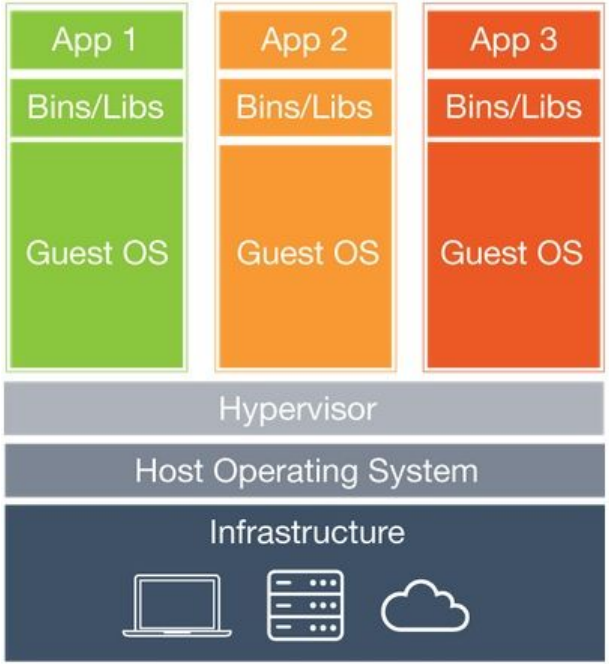
```
docker container run debian echo 'Hola Mundo!'
```

¿Qué pasó acá?

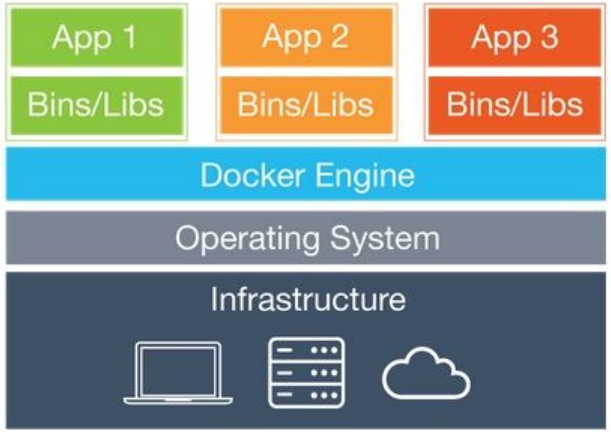
```
docker container run debian echo 'Hola Mundo!'
```

- `docker` — Comando
- `container run` — Sub-Comando
- `debian` — Imagen
- `echo 'Hola Mundo!'` — Comando a correr dentro de la imagen

¿Qué pasó acá? (Por detrás)



Virtual Machines



Containers

Segundo ejercicio

Segundo ejercicio

```
docker container run -ti debian bash
```

Terminal
Interactiva

Segundo ejercicio

Dentro del container:

```
cat /etc/os-release
```

Segundo ejercicio

Dentro del container:

```
cat /etc/os-release

apt-get update && apt-get upgrade -y
```

Tercer ejercicio

Tercer ejercicio

Dentro del container:

```
apt-get install nginx
```

Tercer ejercicio

Dentro del container:

```
apt-get install -y nginx

nginx
```

Tercer ejercicio

Dentro del container:

```
apt-get install -y nginx
```

```
nginx
```

```
apt-get install -y curl
```

Tercer ejercicio

Dentro del container:

```
apt-get install -y nginx

nginx

apt-get install -y curl

curl localhost:80
```

Tercer ejercicio

En tu máquina:

curl localhost:80 o abrir con browser

Cuarto ejercicio

Cuarto ejercicio

```
docker container run -ti -p 8080:80 \
   debian bash
```

Cuarto ejercicio

```
docker container run -ti -p 8080:80 \
   mercadolibre/nginx-workshop bash
```

Cuarto ejercicio

Dentro del container:

```
cat /etc/os-release

apt-get update && apt-get upgrade -y

apt-get install nginx curl -y

nginx

curl localhost:80
```

Cuarto ejercicio

En tu máquina:

curl localhost:8080 o abrir con browser

Quinto ejercicio

Quinto ejercicio

```
docker container run -p 8080:80 \
   -d \
   mercadolibre/nginx-workshop \
   nginx
```

Quinto ejercicio

```
docker container ps

docker container ps -a
```

Quinto ejercicio

```
docker container run -d \
   mercadolibre/nginx-workshop \
   bash -c “sleep infinity &”
```

Quinto ejercicio

```
docker container run -p 8080:80 \
   -d \
   mercadolibre/nginx-workshop \
   nginx -g “daemon off;”
```

Quinto ejercicio

```
docker container stop CONTAINER_ID/NAME

docker container kill CONTAINER_ID/NAME
```

Quinto ejercicio

```
docker container run -p 8080:80 \
   --name nginx \
   -d \
   mercadolibre/nginx-workshop \
   nginx -g “daemon off;”
```

Quinto ejercicio

```
<html>
  <body>
    Hello world!
  </body>
</html>
```

Quinto ejercicio

```
docker container cp \
   index.html nginx:/myhtmls/index.html
```

SOURCE:PATH

DESTINATION:PATH

Quinto ejercicio

```
docker container run -p 8080:80 -d \
   --name nginx \
   -v /home/workshop/html:/myhtmls \
   mercadolibre/nginx-workshop nginx
```

Quinto ejercicio

```
docker container run -p 8080:80 -d \
   --name nginx \
   --mount “type=bind,\
      source=/home/workshop/html,\
      target=/myhtmls” \
   mercadolibre/nginx-workshop nginx
```

Sexto ejercicio

Sexto ejercicio

```
docker container run -ti -m 64m \
   --name memory \
   ubuntu dd bs=250M if=/dev/zero of=/dev/null
```

Sexto ejercicio

```
docker container stats memory
```

Sexto ejercicio

```
docker container run -ti -m 64m \
   --name memory-2 \
   mercadolibre/stress:latest \
      --vm 1 --vm-bytes 128M
```

Sexto ejercicio

```
docker container run -ti --cpus 1 \
   --name cpu \
   mercadolibre/stress:latest \
      --cpu 10
```

Sexto ejercicio

```
docker container run -ti --cpus 0.5 \
   --name cpu-2 \
   mercadolibre/stress:latest \
      --cpu 10
```

Sexto ejercicio

```
docker container run -ti --cpuset-cpus 0-1 \
   --name cpu-3 \
   mercadolibre/stress:latest \
      --cpu 10
```

Séptimo ejercicio

Séptimo ejercicio

https://hub.docker.com/explore/

Séptimo ejercicio

Java -> https://hub.docker.com/_/openjdk

Python -> https://hub.docker.com/_/python

Go -> https://hub.docker.com/_/golang/

NodeJS -> https://hub.docker.com/_/node

Día 2

Agenda

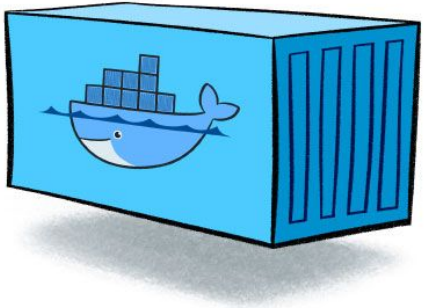
Día 1 - ¿Qué es Docker? - Conceptos
"Hola Mundo"
Administración a fondo de containers

Día 2 - Todo lo que hay que saber de imágenes

Día 3 - Orquestación de distintos componentes
Comandos de troubleshooting
Manejo de networking y volúmenes
Monitoreo

Día 4 - El Mundo de Producción
Lecciones aprendidas
Cómo Meli utiliza Docker

Container vs Image



Primer ejercicio

Primer ejercicio

```
docker container run -ti debian bash
```

Primer ejercicio

```
Dentro del container:

apt-get update

apt-get install -y nginx curl
```

Primer ejercicio

```
docker container ps

docker container ps -a
```

Primer ejercicio

```
docker container commit ID/NAME
```

Primer ejercicio

```
docker image tag ID NAME[:TAG]
```

Primer ejercicio

```
docker image tag ID my-nginx
```

Primer ejercicio

```
docker container run -ti my-nginx bash
```

Segundo ejercicio

Segundo ejercicio

```
docker container run -ti --name nginx \
   my-nginx bash
```

Segundo ejercicio

```
docker container rm ID/NAME
```

Segundo ejercicio

```
<html>
  <body>
    Hello world!
  </body>
</html>
```

Segundo ejercicio

```
docker container cp \
   index.html nginx:/var/www/html/index.html
```

Segundo ejercicio

```
docker container diff ID/NAME
```

Les resulta conocido?


git

Segundo ejercicio

```
docker image history my-nginx
```

Tercer ejercicio

Tercer ejercicio

```
docker container run --name paso-1 \
   debian apt-get update
```

Tercer ejercicio

```
docker container commit paso-1 paso-1
```

Tercer ejercicio

```
docker container run --name paso-2 \
   paso-1 apt-get install -y nginx curl
```

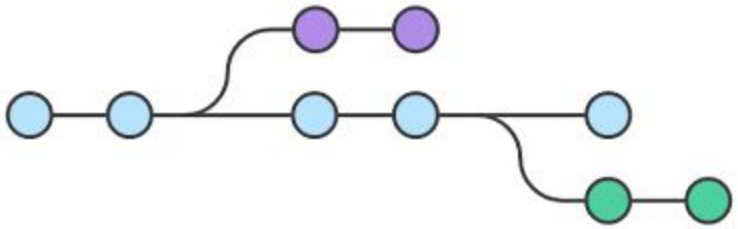
Tercer ejercicio

```
docker container commit paso-2 my-nginx
```

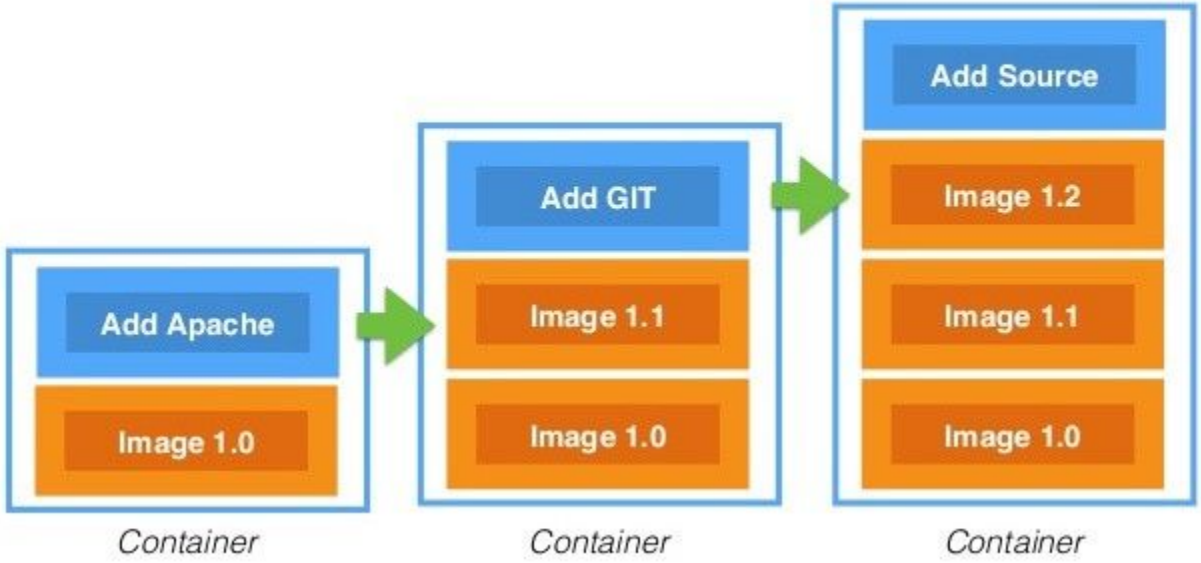
Tercer ejercicio

```
docker image history my-nginx
```

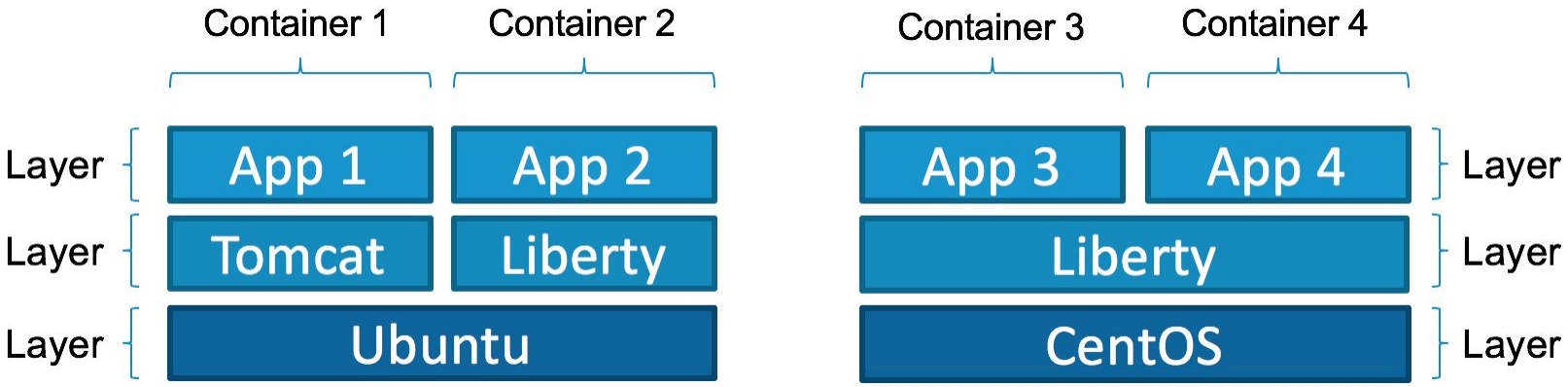
Tercer ejercicio



Tercer ejercicio



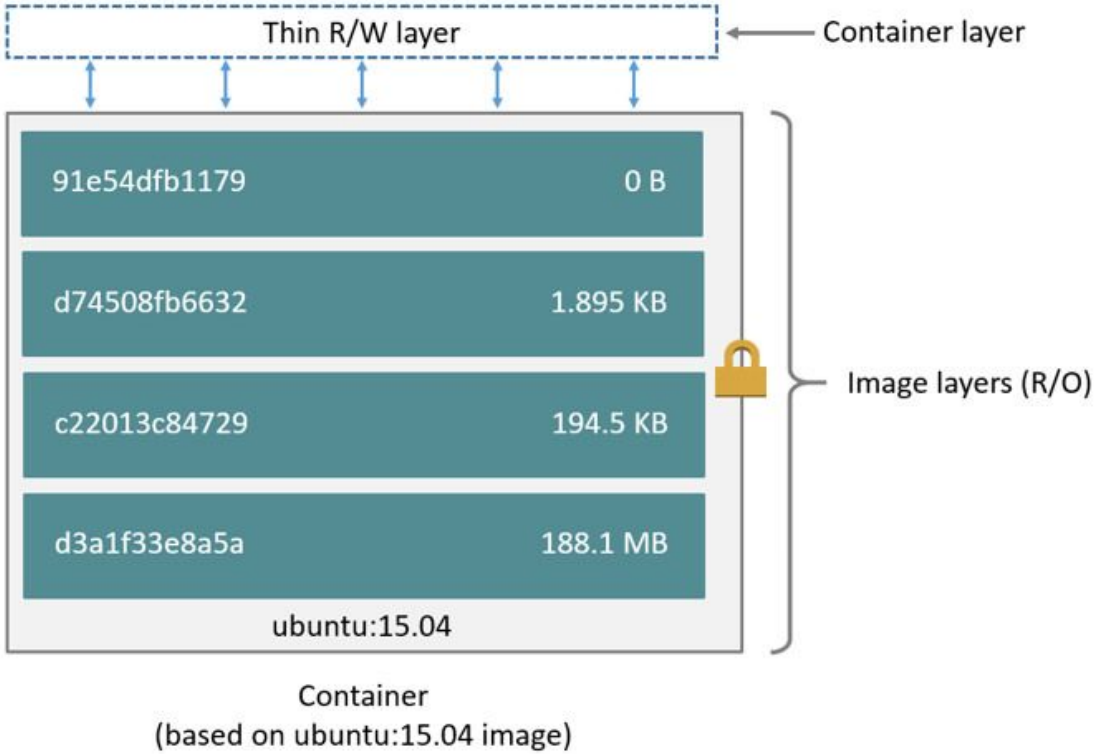
Tercer ejercicio



Tercer ejercicio

```
docker image inspect debian
docker image inspect paso-1
docker image inspect paso-2
```

Tercer ejercicio



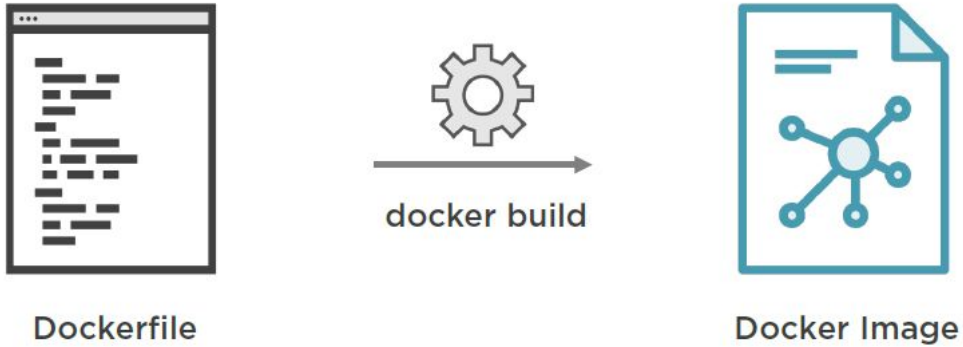
Tercer ejercicio

- Un container es simplemente una **capa fina de lectura/escritura**.
  - Imágenes base no son copiadas a los containers

- Copy-on-Write (**CoW**)
  - Cuando un **archivo de la imagen base** cambia:
    - Se **copia** el archivo a la capa de lectura/escritura
    - Luego, se **modifica** el archivo copiado

Cuarto ejercicio

Dockerfiles


docker build
Dockerfile
Docker Image

Instrucciones

```
FROM            # imagen que extiendo (necesario)

COPY            # copiar un archivo del contexto a la imagen
ADD             # similar a COPY (es preferible usar COPY)
RUN             # correr un comando en la imagen durante el buildeo (instalar algo)
ENV             # declarar variables de entorno
WORKDIR         # carpeta default en la que va a correr todo
ENTRYPOINT      # script default que va a correr la imagen
CMD             # comando default que va a correr la imagen (parámetros del entry)
ONBUILD         # cuando una imagen extienda de esta lo primero que se va a correr
VOLUME          # declarar un volumen
EXPOSE          # declarar que exponemos un puerto
```

Documentacion: https://docs.docker.com/engine/reference/builder/

Cuarto ejercicio

```
docker image build -t NAME CONTEXT_PATH
docker image build -t NAME .
```

Armemos un Dockerfile

Cuarto ejercicio

```
FROM debian:latest

RUN apt-get update
RUN apt-get install -y net-tools

ENTRYPOINT ["/bin/ping"]
```

Cuarto ejercicio

```
docker image build -t myping .

docker container run -ti myping \
   mercadolibre.com
```

```
Sending build context to Docker daemon  14.34kB
Step 1/4 : FROM debian:latest
 ---> 8626492fecd3
Step 2/4 : RUN apt-get update
 ---> Running in 3e426225e592
...
Removing intermediate container 3e426225e592
 ---> 255c44dba14a
Step 3/4 : RUN apt-get install -y net-tools
 ---> Running in 334b8eed50d5
...
Removing intermediate container 334b8eed50d5
 ---> 4220f11be34f
Step 4/4 : ENTRYPOINT ["/bin/ping"]
 ---> Running in 082247be3f2d
Removing intermediate container 082247be3f2d
 ---> 9669c18caef1
Successfully built 9669c18caef1
Successfully tagged myping:latest
```

```
Sending build context to Docker daemon  14.34kB
Step 1/4 : FROM debian:latest
 ---> 8626492fecd3
Step 2/4 : RUN apt-get update
 ---> Using cache
 ---> 255c44dba14a
Step 3/4 : RUN apt-get install -y net-tools
 ---> Using cache
 ---> 4220f11be34f
Step 4/4 : ENTRYPOINT ["/bin/ping"]
 ---> Using cache
 ---> 9669c18caef1
Successfully built 9669c18caef1
Successfully tagged myping:latest
```

Cuarto ejercicio

```
FROM debian:latest

RUN apt-get update
RUN apt-get install -y net-tools

ENTRYPOINT ["/bin/ping"]
CMD ["mercadolibre.com"]
```

Cuarto ejercicio

```
docker image build -t myping .
docker container run -ti myping
```

Quinto ejercicio

Quinto ejercicio

```
FROM ubuntu:latest

RUN apt-get update
RUN apt-get install --no-install-recommends -y nodejs npm ca-certificates
RUN mkdir /app
WORKDIR /app
COPY . /app
RUN npm install

EXPOSE 8080

CMD ["npm", "start"]
```

Quinto ejercicio

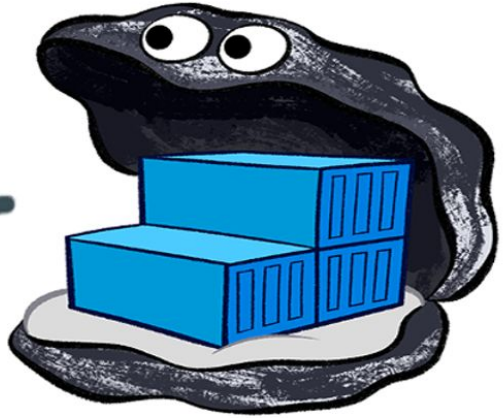
```
docker image build -t myapp .

docker container run -p 8080:8080 -ti myapp
```

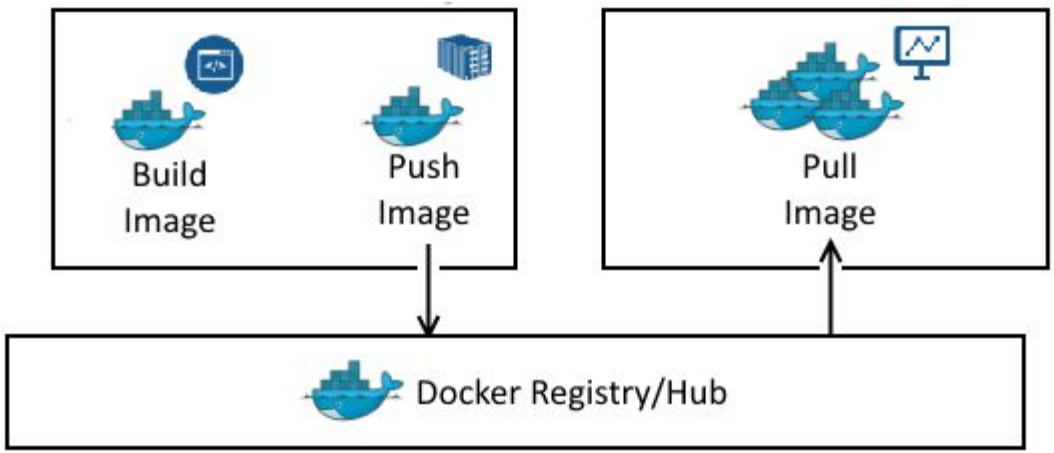
Que pasa si cambiamos algo?

Quinto ejercicio

```
FROM ubuntu:latest

RUN apt-get update
RUN apt-get install --no-install-recommends -y nodejs npm ca-certificates
RUN mkdir /app
WORKDIR /app
COPY package.json package-lock.json /app
RUN npm install
COPY . /app

EXPOSE 8080

CMD ["npm", "start"]
```

Cómo comparto?

docker  REGISTRY



https://hub.docker.com/

Quinto ejercicio

```
docker image build -t USERNAME/IMAGE:TAG .
docker image push USERNAME/IMAGE:TAG
docker image pull USERNAME/IMAGE:TAG
```

Sexto ejercicio

Hagamos una imagen base

Sexto ejercicio

```
FROM ubuntu:latest

RUN apt-get update
RUN apt-get install --no-install-recommends -y nodejs npm ca-certificates
RUN mkdir /app
WORKDIR /app
ONBUILD COPY package.json package-lock.json /app
ONBUILD RUN npm install
ONBUILD COPY . /app

EXPOSE 8080

CMD ["npm", "start"]
```

Sexto ejercicio

```
docker image build -t USERNAME/nodejs:latest .
```

Sexto ejercicio

```
FROM USERNAME/nodejs:latest

# Cri cri
# Con tener un dockerfile que solamente extienda de del anterior ya funcionaria
# Si fuese necesario podría hacer más cosas
```

Sexto ejercicio

```
docker image build -t USERNAME/myapp:latest .
```

Séptimo ejercicio

Séptimo ejercicio

```
FROM ubuntu:latest

RUN apt-get update
RUN apt-get install -y gcc
RUN mkdir /app
WORKDIR /app
COPY main.c /app
RUN gcc -o main main.c

CMD ["/app/main"]
```

Séptimo ejercicio

```
docker image build -t mycapp .
```

Steps

Steps

```
FROM ubuntu:latest

RUN apt-get update
RUN apt-get install -y gcc
RUN mkdir /app
WORKDIR /app
COPY main.c /app
RUN gcc -o main main.c

FROM progrium/busybox
COPY --from=0 /app/main /bin/main
CMD ["/bin/main"]
```

Steps

```
docker image build -t mycapp-2 .
```

Steps

```
docker image ls -t mycapp-2 | grep mycapp
```

Steps

```
mycapp-2                    latest          3db6df8308f1        50 seconds ago      4.8MB
mycapp                      latest          1fec6c19b3a4        6 minutes ago       229MB
```

Tips

1- Pensar bien la imagen base necesito

```
FROM ubuntu:latest

RUN apt-get update
RUN apt-get install --no-install-recommends -y nodejs npm ca-certificates
RUN mkdir /app
WORKDIR /app
COPY . /app
RUN npm install

EXPOSE 8080

CMD ["npm", "start"]
```

1- Pensar bien la imagen base necesito

```
lbrizuela@lbrizuela:~$ docker images debian
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
debian              latest              8626492fecd3        2 weeks ago         101MB
lbrizuela@lbrizuela:~$ docker images ubuntu
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ubuntu              latest              452a96d81c30        2 weeks ago         79.6MB
lbrizuela@lbrizuela:~$ docker images alpine
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              3fd9065eaf02        4 months ago        4.15MB
```

```
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
ruby                2.3-alpine          f7980887bc4b        6 weeks ago         115MB
```

2- Compartir lo básico

```
FROM debian:latest

RUN apt-get update && apt-get install -y net-tools
RUN apt-get install SUPER_PARCHE_DE_SEGURIDAD

ENTRYPOINT ["/bin/ping"]
```

```
FROM debian:latest

RUN apt-get update && apt-get install -y nginx curl
RUN apt-get install SUPER_PARCHE_DE_SEGURIDAD

COPY index.html /var/www/html/index.html
CMD ["nginx", "-g", "daemon off;"]
```

2- Compartir lo básico

```
FROM debian:latest

RUN apt-get update && apt-get install SUPER_PARCHE_DE_SEGURIDAD
```

```
FROM my-base:latest

RUN apt-get install -y net-tools

ENTRYPOINT ["/bin/ping"]
```

3- El orden importa

```
FROM debian:latest

RUN apt-get update && apt-get install -y nginx curl

COPY index.html /var/www/html/index.html

CMD ["nginx", "-g", "daemon off;"]
```

3- El orden importa

```
Sending build context to Docker daemon   2.56kB
Step 1/3 : FROM debian:latest
 ---> 8626492fecd3
Step 2/3 : RUN apt-get update && apt-get install -y nginx curl
 ---> Using cache
 ---> a2519700d9f4
Step 3/3 : COPY index.html /var/www/html/index.html
 ---> Using cache
 ---> 5a9f2802544c
Successfully built 5a9f2802544c
Successfully tagged my-nginx:latest
```

3- El orden importa

```
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM debian:latest
 ---> 8626492fecd3
Step 2/3 : RUN apt-get update && apt-get install -y nginx curl
 ---> Using cache
 ---> a2519700d9f4
Step 3/3 : COPY index.html /var/www/html/index.html
 ---> 110ac8f890d9
Removing intermediate container 14db3fb42a7e
Successfully built 110ac8f890d9
Successfully tagged my-nginx:latest
```

3- El orden importa

```
FROM debian:latest

COPY index.html /var/www/html/index.html

RUN apt-get update && apt-get install -y nginx curl

CMD ["nginx", "-g", "daemon off;"]
```

3- El orden importa

```
Sending build context to Docker daemon  3.072kB
Step 1/3 : FROM debian:latest
 ---> 8626492fecd3
Step 2/3 : COPY index.html /var/www/html/index.html
 ---> 4113ad414410
Removing intermediate container 561470029d03
Step 3/3 : RUN apt-get update && apt-get install -y nginx curl
 ---> Running in ab9bbd30a6ce
Get:1 http://security.debian.org/debian-security stretch/updates InRelease [94.3 kB]
Ign:2 http://cdn-fastly.deb.debian.org/debian stretch InRelease
Get:3 http://cdn-fastly.deb.debian.org/debian stretch-updates InRelease [91.0 kB]
Get:4 http://cdn-fastly.deb.debian.org/debian stretch Release [118 kB]
```

4- Cuidado como correr “apt-get update”

```
FROM debian:latest

RUN apt-get update
RUN apt-get install -y net-tools

ENTRYPOINT ["/bin/ping"]
```

4- Cuidado como correr “apt-get update”

```
FROM debian:latest

RUN apt-get update
RUN apt-get install -y net-tools ALGO_NUEVO

ENTRYPOINT ["/bin/ping"]
```

```
docker image build --no-cache -t myapp .
```

4- Cuidado como correr “apt-get update”

```
FROM debian:latest

RUN apt-get update && apt-get install -y net-tools ALGO_NUEVO

ENTRYPOINT ["/bin/ping"]
```

5- Organiza tus imágenes con labels

```
FROM debian:latest

LABEL app=pepito

RUN apt-get update && apt-get install -y net-tools ALGO_NUEVO

ENTRYPOINT ["/bin/ping"]
```

6- Variables de entorno

```
FROM ruby:2.3-alpine

ENV RAILS_ENV=production \
  SOMETHING=something

ENTRYPOINT ["/app/run.sh"]
```

Día 3

Agenda

Día 1 - ¿Qué es Docker? - Conceptos
"Hola Mundo"
Administración a fondo de containers

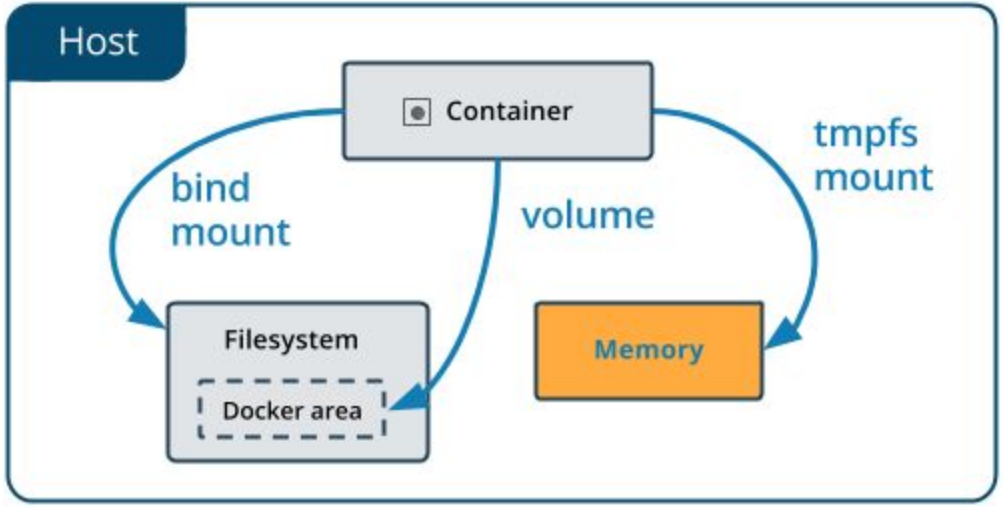
Día 2 - Todo lo que hay que saber de imágenes

Día 3 - Manejo de networking y volúmenes
Orquestación de distintos componentes
Comandos de troubleshooting

Día 4 - El Mundo de Producción
Lecciones aprendidas
Cómo Meli utiliza Docker

Volúmenes

Volúmenes


Host
Container
bind
mount
volume
tmpfs
mount
Filesystem
Docker area
Memory

Volúmenes

```
docker container run -p 8080:80 -d \
   --name nginx \
   --mount “type=bind,\
      source=/home/workshop/html,\
      target=/myhtmls” \
   mercadolibre/nginx-workshop nginx
```

Volúmenes

```
docker container run -p 8080:80 -d \
   --name nginx \
   --mount “type=tmpfs,\
      source=/home/workshop/html,\
      target=/myhtmls,
      tmpfs-size=2G” \
   mercadolibre/nginx-workshop nginx
```

Volúmenes

```
docker volume create NAME
```

Volúmenes

```
docker volume create test
```

Volúmenes

```
docker container run -p 8080:80 -d \
   --name nginx \
   --mount “type=volume,\
      source=test,\
      target=/myhtmls” \
   mercadolibre/nginx-workshop nginx
```

Volúmenes

```
docker volume inspect test
```

Volúmenes

```
[
    {
        "CreatedAt": "2018-05-22T15:15:47-03:00",
        "Driver": "local",
        "Labels": {},
        "Mountpoint": "/var/lib/docker/volumes/test/_data",
        "Name": "test",
        "Options": {},
        "Scope": "local"
    }
]
```

Volúmenes

```
docker volume create \
   --driver local
   -o type=tmpfs
   -o device=tmpfs
   -o o=size=2G
   test
```

Volúmenes

```
docker volume create \
   --driver local
   -o type=nfs
   -o device=:/path
   -o o=addr=192.168.1.1,rw
   test
```

https://store.docker.com/search?type=plugin&category=volume

Primer ejercicio

Primer ejercicio

```
docker container run -p 5000:5000 -d \
   --name app \
   mercadolibre/nodeapp-workshop
```

Primer ejercicio

```
docker container run -d \
   --name redis \
   redis:alpine --appendonly yes
```

Primer ejercicio

```
docker container inspect redis
```

Primer ejercicio

```
docker volume inspect ID
```

Primer ejercicio

```
docker container run -p 5000:5000 -d \
   -e REDIS_HOST=IP_ADDRESS
   mercadolibre/nodeapp-redis-workshop
```

Primer ejercicio

```
docker container restart redis redis-app
```

Segundo ejercicio

Segundo ejercicio

```
docker container run -d \
   --name redis \
   redis:alpine --appendonly yes
```

Segundo ejercicio

```
docker volume create redis_data
```

Segundo ejercicio

```
docker container run -p 6380:6380 -d \
   --name redis \
   --mount “type=volume,\
      source=test,\
      target=/data” \
   redis:alpine --appendonly yes
```

Storage drivers

Config storage driver

/etc/docker/daemon.json

```
{
    "storage-driver": "overlay2"
}
```

Redes

Tercer ejercicio

Tercer ejercicio

```
docker container run --net=host -d \
   --name redis \
   redis:alpine --appendonly yes
```

Tercer ejercicio

```
docker container run --net=host -d \
   -e REDIS_HOST=localhost
   mercadolibre/nodeapp-redis-workshop
```

Cuarto ejercicio

Cuarto ejercicio

```
docker network create my-net
```

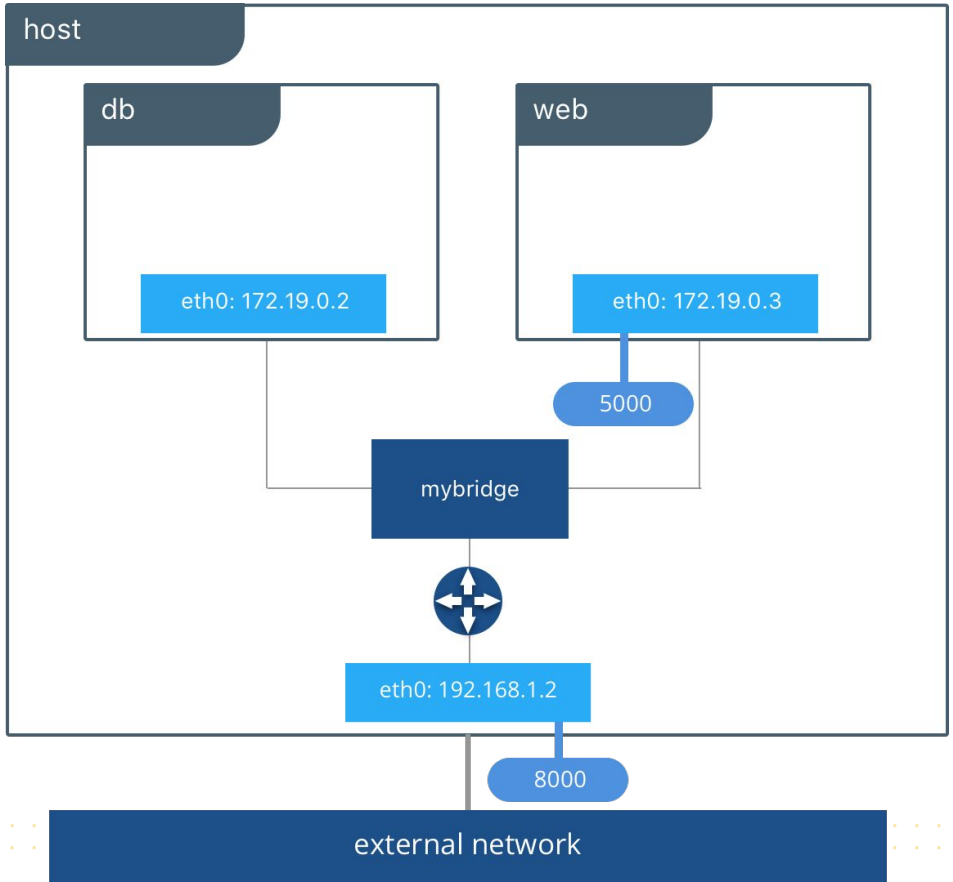
Cuarto ejercicio

```
docker container run --net=my-net -d \
   --name redis \
   redis:alpine --appendonly yes
```

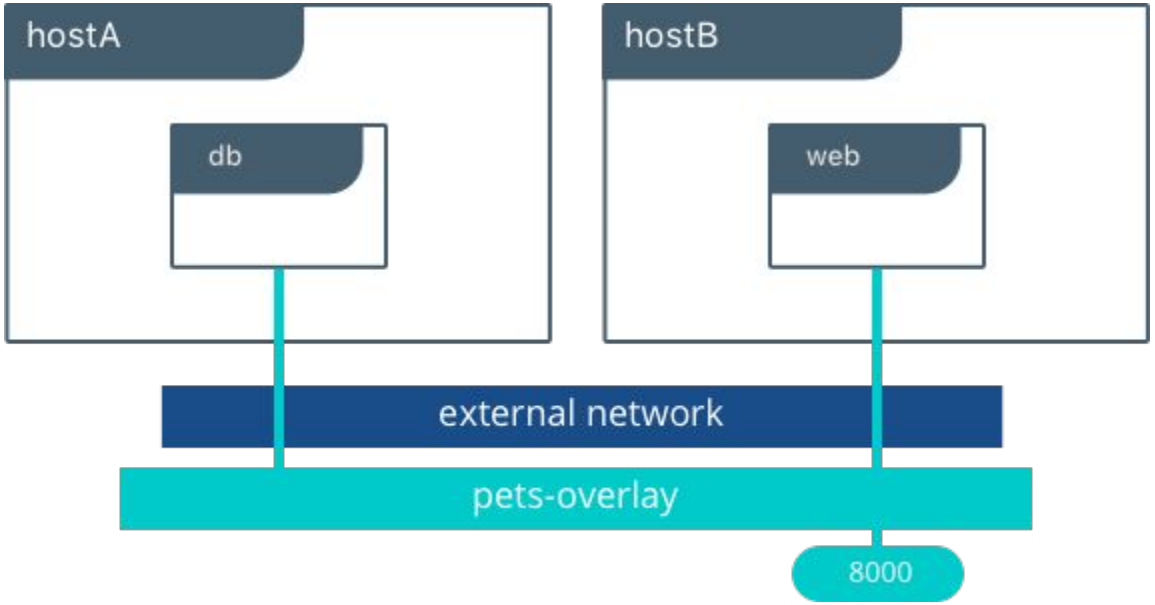
Cuarto ejercicio

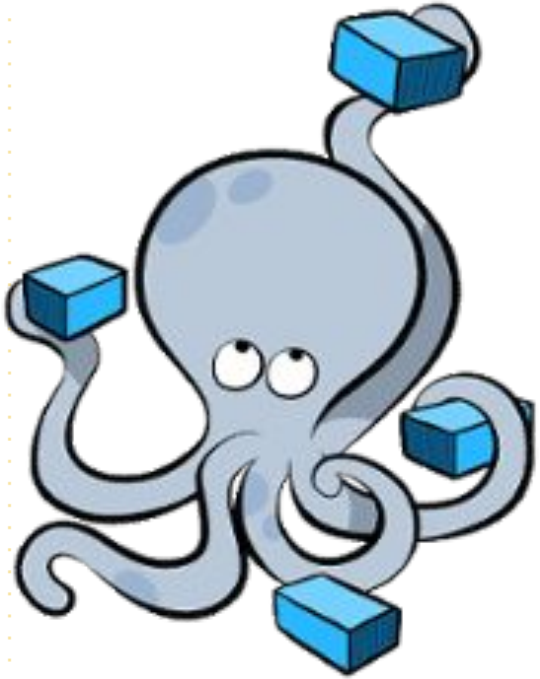
```
docker container run --net=my-net -d \
   -p 5000:5000 \
   --name redis-app \
   -e REDIS_HOST=redis \
   mercadolibre/nodeapp-redis-workshop
```

Redes


host
db
web
eth0: 172.19.0.2
eth0: 172.19.0.3
5000
mybridge
eth0: 192.168.1.2
8000
external network

Redes





Docker Compose

https://docs.docker.com/compose/install/

Quinto ejercicio

docker-compose.yml

```yaml
version: '3'
services:
  web:
    container_name: myapp
    image: mercadolibre/nodeapp-redis-workshop
    ports:
      - "5000:5000"
    environment:
      REDIS_HOST: redis
    depends_on:
       - redis

  redis:
    image: redis:alpine
    command: --appendonly yes
    volumes:
      - ./data:/data
```

Docker Compose

```
docker-compose up -d

docker-compose restart myapp

docker-compose down
```

Troubleshooting

(No tengo idea qué le pasa a mi app)

Logs

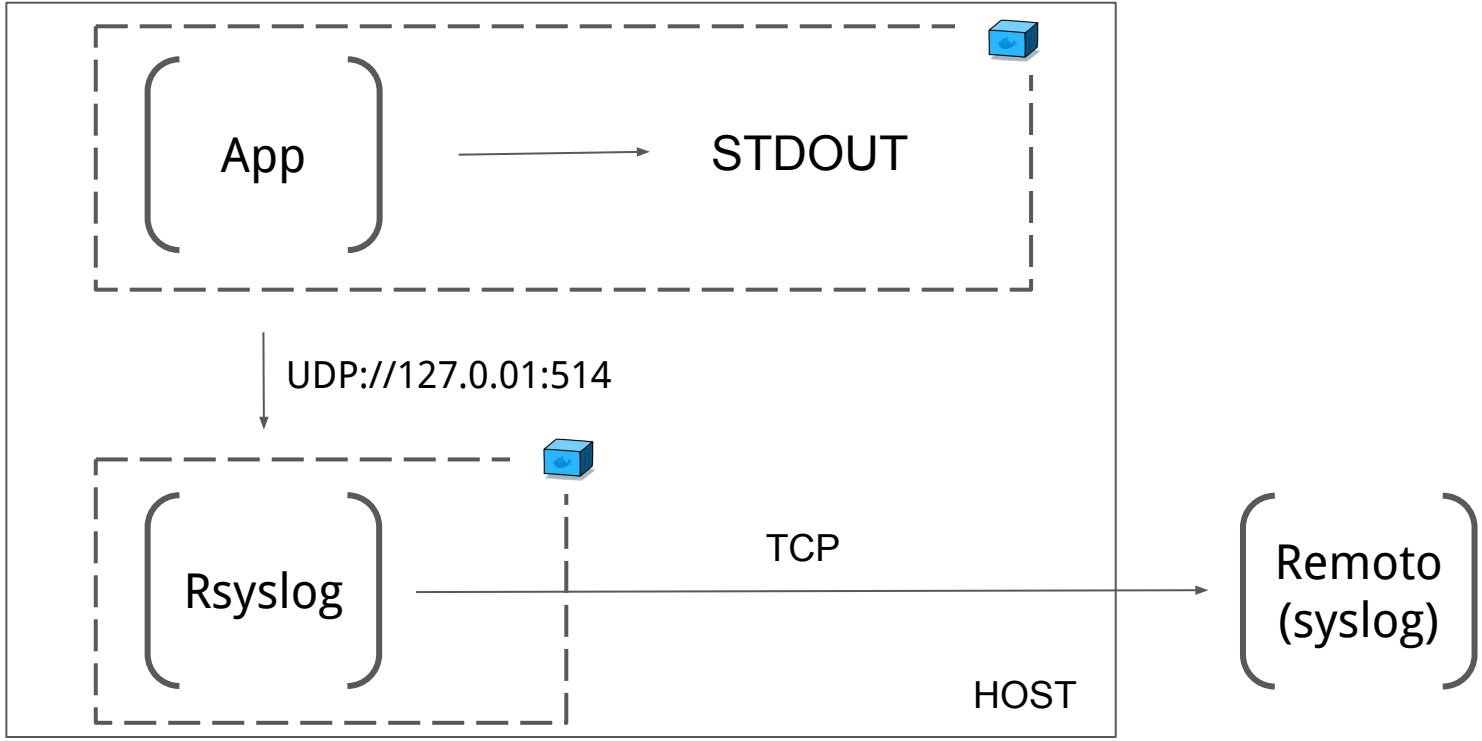
```
docker container run \
    -d debian bash -c \
        'for i in {1..100}; do echo "$i" ; done'

docker logs ID
```

Logs

```
docker container run \
   --log-driver syslog
   --log-opts syslog-address [udp|tcp]://host:port
   -d debian bash -c \
      'for i in {1..100}; do echo "$i" ; done'
```

https://docs.docker.com/config/containers/logging/configure/#supported-logging-drivers

Logs



Troubleshooting

(No tengo idea qué hace este container)

Troubleshooting

```
docker exec -ti ID COMMAND

docker inspect ID

docker diff ID
```

Troubleshooting

(No tengo idea que está pasando con docker)

Troubleshooting

```
docker info

systemctl status docker
```

Día 4

Agenda

Día 1 - ¿Qué es Docker? - Conceptos
"Hola Mundo"
Administración a fondo de containers

Día 2 - Todo lo que hay que saber de imágenes

Día 3 - Manejo de networking y volúmenes
Orquestación de distintos componentes
Comandos de troubleshooting

Día 4 - El Mundo de Producción
Lecciones aprendidas
Cómo Meli utiliza Docker

Orquestadores

Orquestadores

Muchos containers

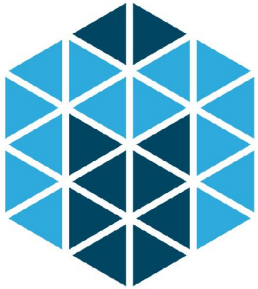
Donde ubicarlos

Compartir recursos eficientemente

Abstraerse de la infraestructura

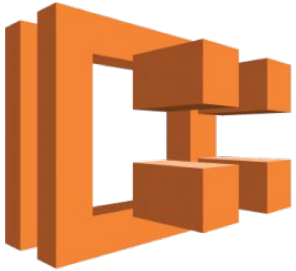
Orquestadores

kubernetes



MESOS

HashiCorp
Nomad



Amazon ECS

Docker Swarm

Sistema sencillo

Integrado con Docker

Se usa el mismo cliente

Pequeña Comunidad

Pequeña escala

Kubernetes

Sistema complejo

Gran comunidad ++

Muchos componentes y plugins

Gran escala

Utilidades para deployar fácilmente

Preferible para aplicaciones Stateless

kubernetes

Mesos

Sistema no tan complejo

Gran comunidad +

Frameworks (Aurora, Marathon, etc)

Gran escala

Buen soporte para aplicaciones Stateful*

Soporte para correr cosas que no sean Containers

Nomad

Sistema sencillo

Pequeña comunidad

Solo scheduler

Gran escala

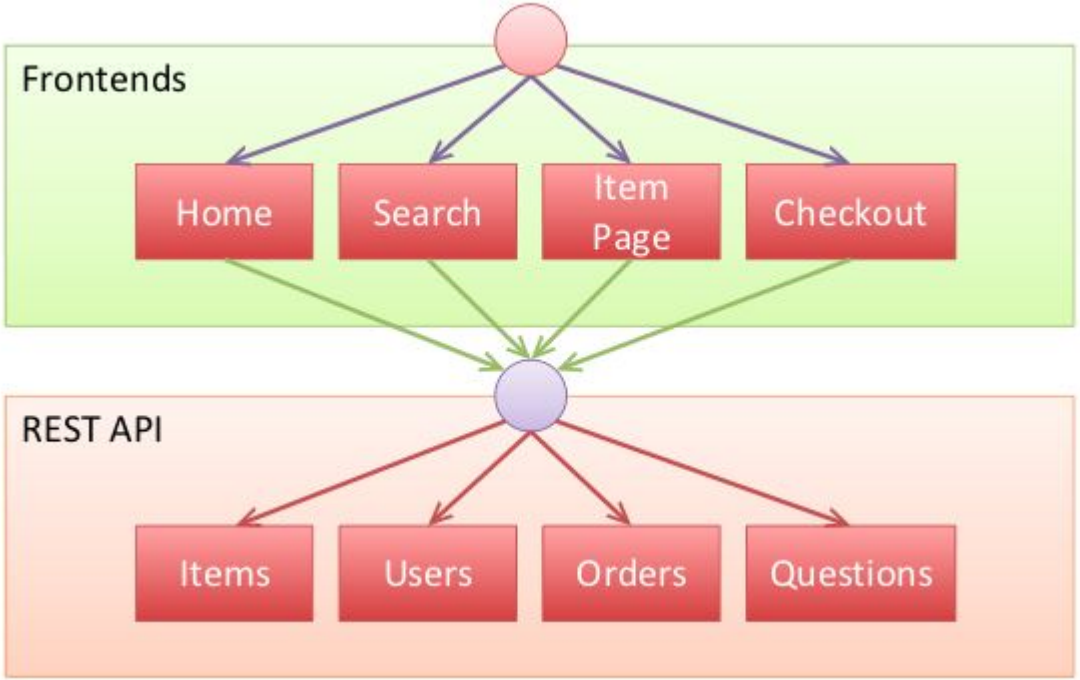
Soporte para correr cosas que no sean Containers

Ejemplo

https://labs.play-with-docker.com/
https://github.com/dockersamples/example-voting-app

Docker en MercadoLibre

Arquitectura de microservicios



Un poco de contexto

2.500 aplicaciones

1.300 desarrolladores

15.000 instancias

1.500 deploys por día

Fury

Desarrollo

- Entornos unificados (sin importar el SO)
- Una única herramienta
- Simplicidad para sumarse a colaborar en un proyecto

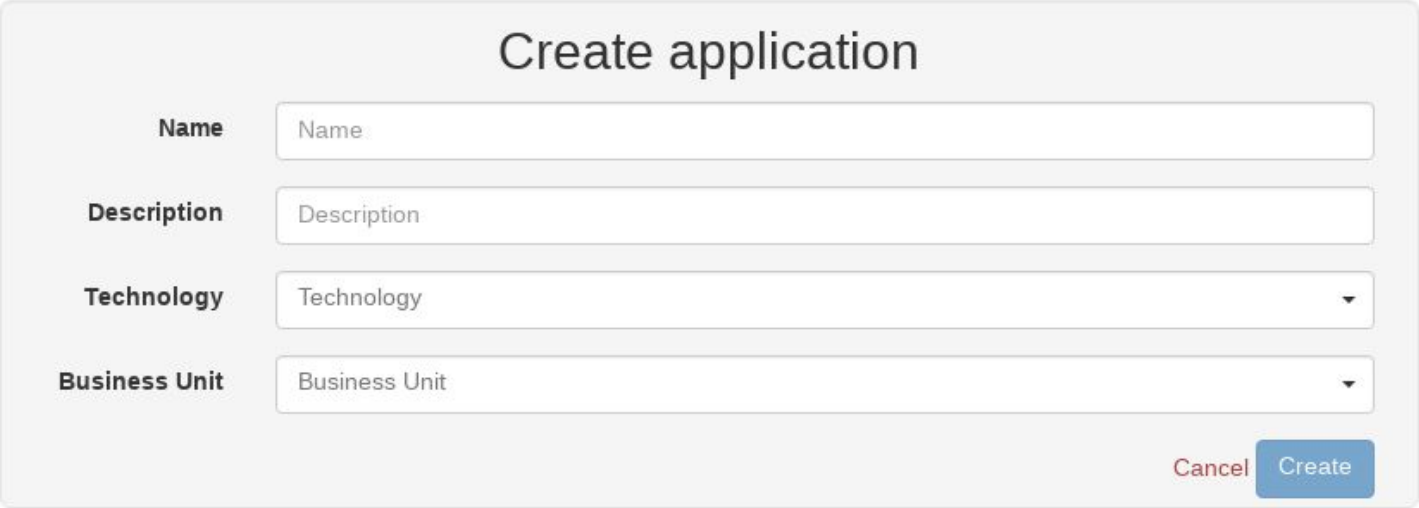
Producción

- Codigo + Entorno en un único lugar
- Imagen inmutable
- No hay necesidad de bootstraps on boot
- Fast build (vs crear un ami)
- Posibilidad de bajar la imagen de producción para debug

Típicos pasos

1. Crear aplicación
2. Desarrollar
3. Crear versión
4. Deployar
5. Troubleshooting (a veces)

1. Crear aplicación



1. Crear aplicación

```
~/c/m/myApp $ tree -a
.
├── Dockerfile
├── Dockerfile.runtime
├── .fury
└── src

1 directory, 3 files
~/c/m/myApp $
```

Dockerfile

```
FROM hub.furycloud.io/mercadolibre/java-maven:jdk8
```

Dockerfile.runtime

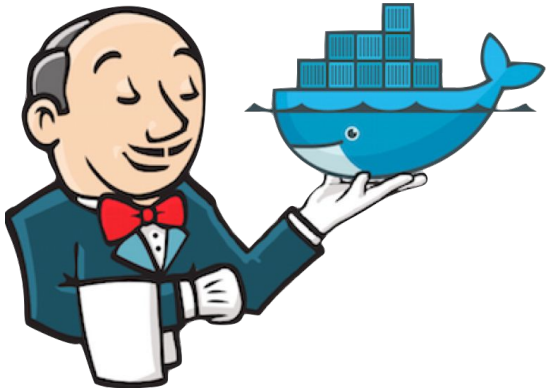
```
FROM hub.furycloud.io/mercadolibre/tomcat:8
```

2. Desarrollar

```
~> fury test
```

```
docker build -t fury-items-api . # Dockerfile
docker run \
  --publish 8080:8080 \
  --publish 8443:8443 \
  --volume $pwd:/app \
  --volume $cache_dir:/cache \
  --env "APPLICATION=items-api" \
  fury-items-api \
  /commands/test.sh
```

3. Crear versión



```
~> fury create-version 0.0.1
```

```
docker build -t imagen-dev . # Dockerfile
docker run -v $pwd:/app -v $cache:/cache /commands/test.sh
docker run -v $pwd:/app -v $cache:/cache -v $tmp:/output /commands/package.sh
cp Dockerfile.runtime $tmp/Dockerfile
cd $tmp
docker build -t imagen-prod . # Dockerfile.runtime
docker push imagen-prod
```

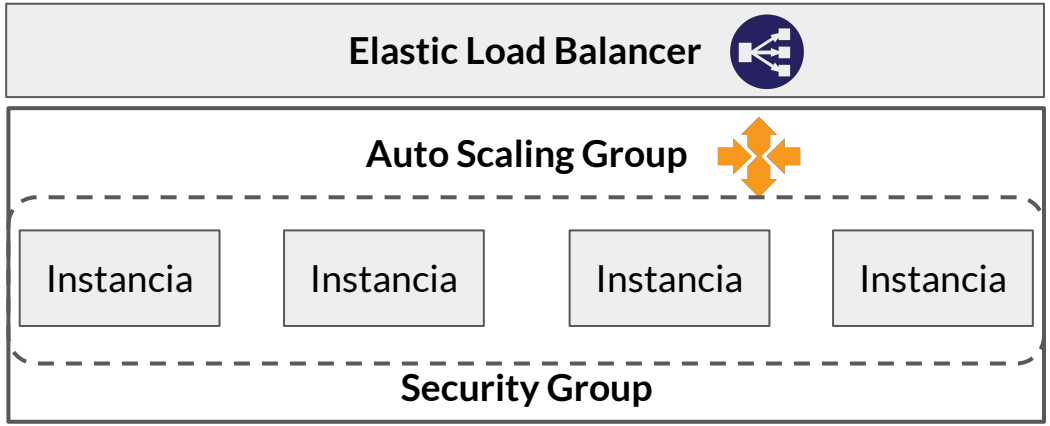
2. Deployar - Estrategias

a. BlueGreen
   i. Recrea infraestructura
   ii. Rollback rápido
   iii. Estrategia por default

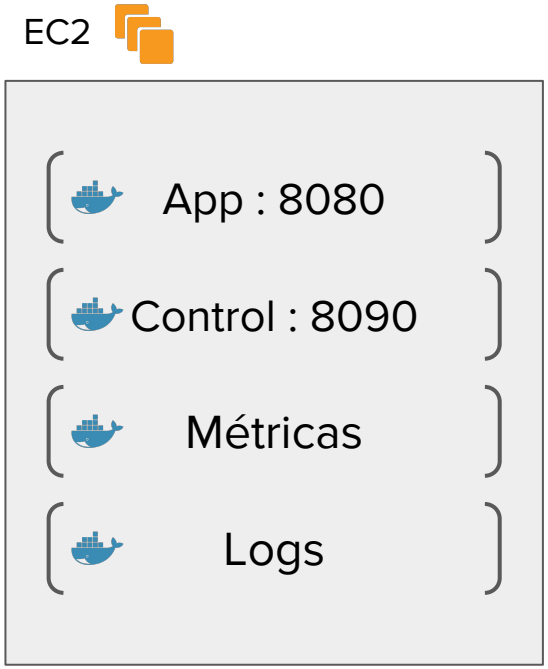
b. Rolling Update
   i. Reutiliza infraestructura
   ii. Deploy más rápido
   iii. Rollback lento

c. Canary
   i. Recrea menos infraestructura
   ii. Se podría decir como un primer paso del bluegreen

2. Deployar - Infraestructura



2. Deployar - Instancia



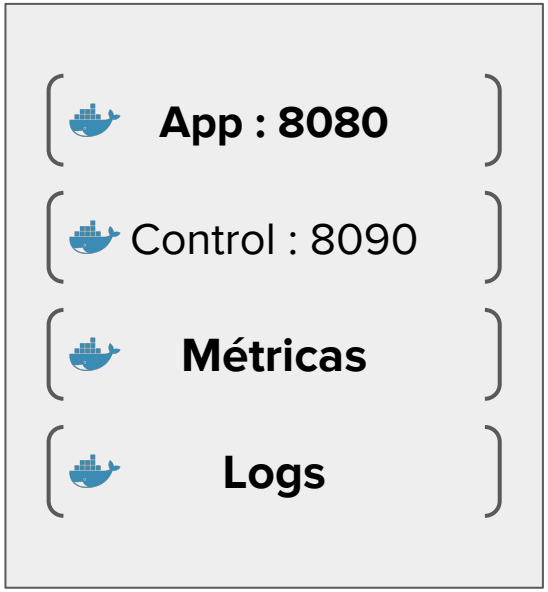
2. Deployar - Instancia

EC2


```
docker pull $REGISTRY/fury-localapi:$LOCALAPI_TAG

docker run -d -i \
      --name localapi \
      --cap-add=NET_ADMIN \
      -e "REGISTRY=$REGISTRY" \
      --restart=always \
      -v /var/run:/var/run \
      -v /root:/root \
      -v /var/log:/var/log \
      -v /tmp/localapi:/tmp \
      -p 8090:8080 \
      --net=host \
      $REGISTRY/fury-localapi:$LOCALAPI_TAG
```

2. Deployar - Instancia

EC2



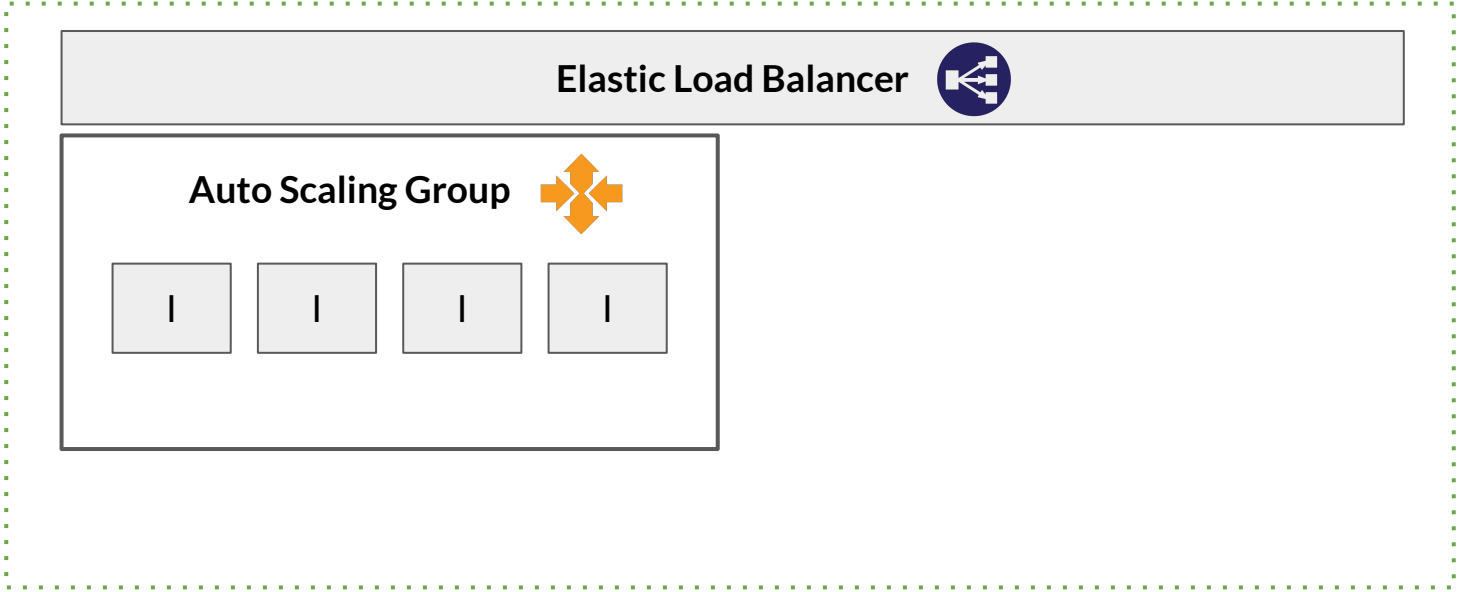

```
docker-compose up -d

application:
      image: $REGISTRI/$APP:$VERSION
      ports:
            - 8080:8080
      mem_limit: ##MEM##
      log_driver: syslog
      log_opt:
            syslog-address: "udp://127.0.0.1:514"
      restart: always

rsyslog:
      ports:
            - 514:514/udp
      ...

datadog:
      ...
```

2. Deployar - Estrategias

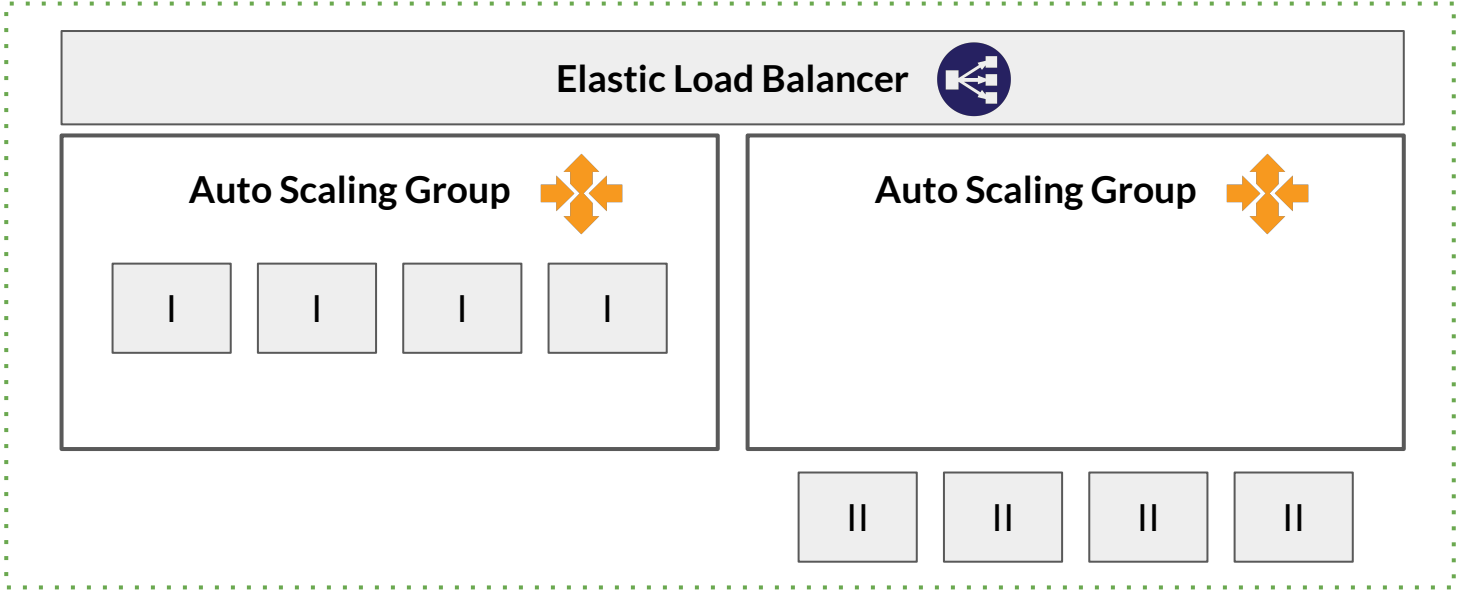
a. BlueGreen
   i. Recrea infraestructura
   ii. Rollback rápido
   iii. Estrategia por default

Upgrade de infraestructura

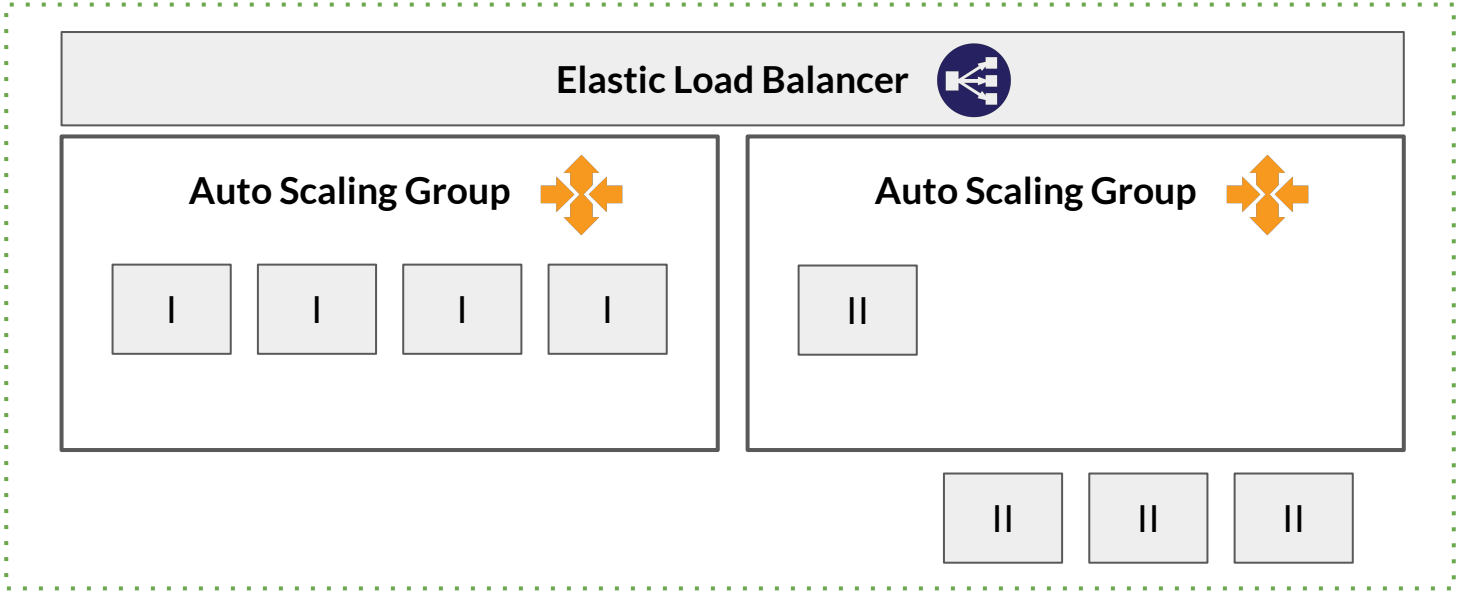
Estrategia Blue Green



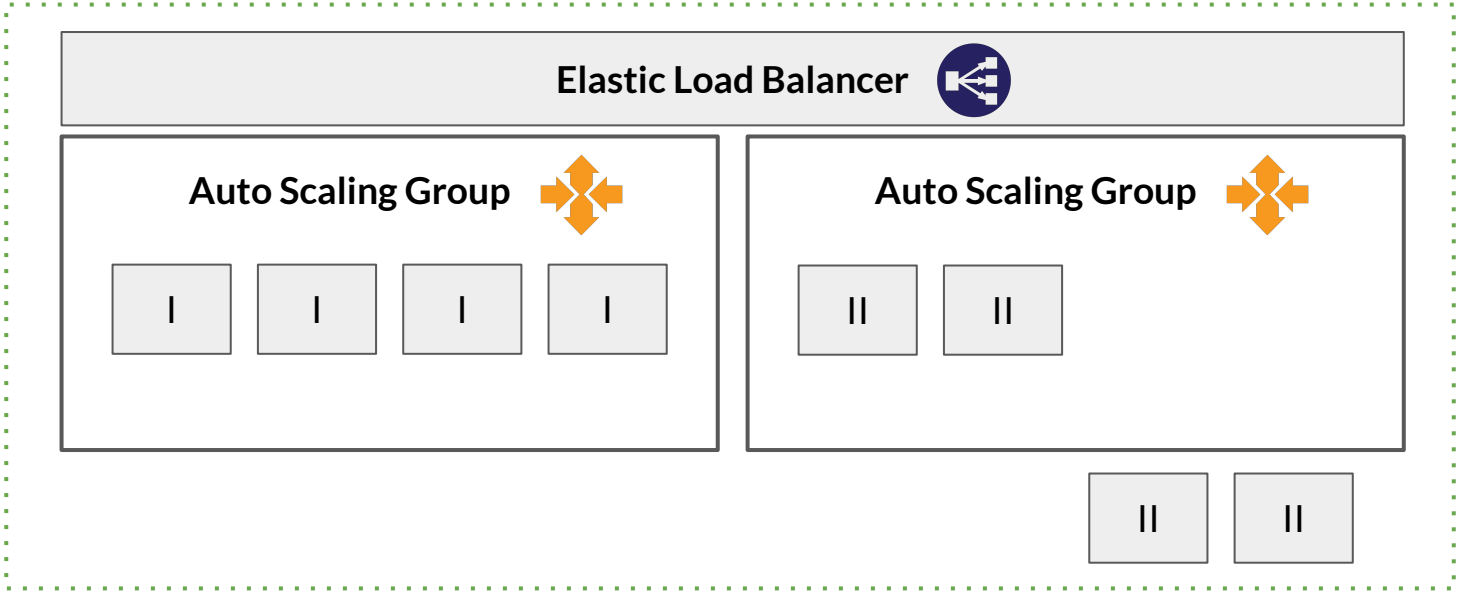
Estrategia Blue Green



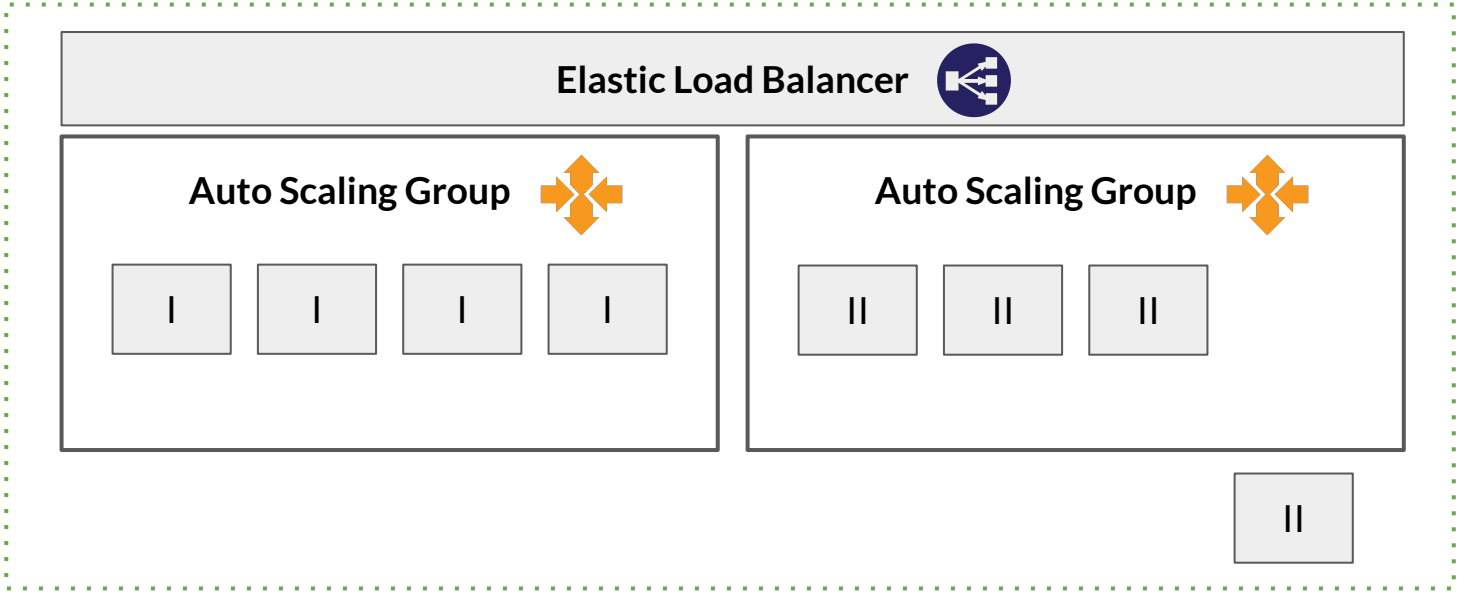
Estrategia Blue Green



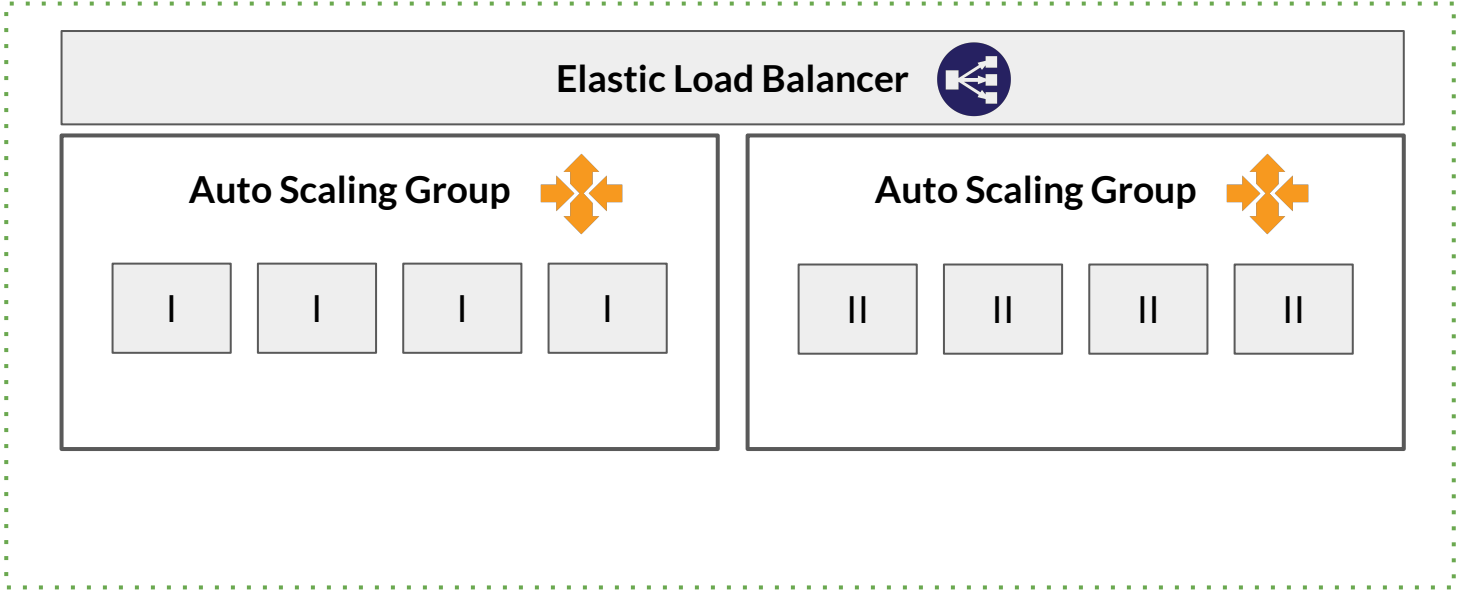
Estrategia Blue Green



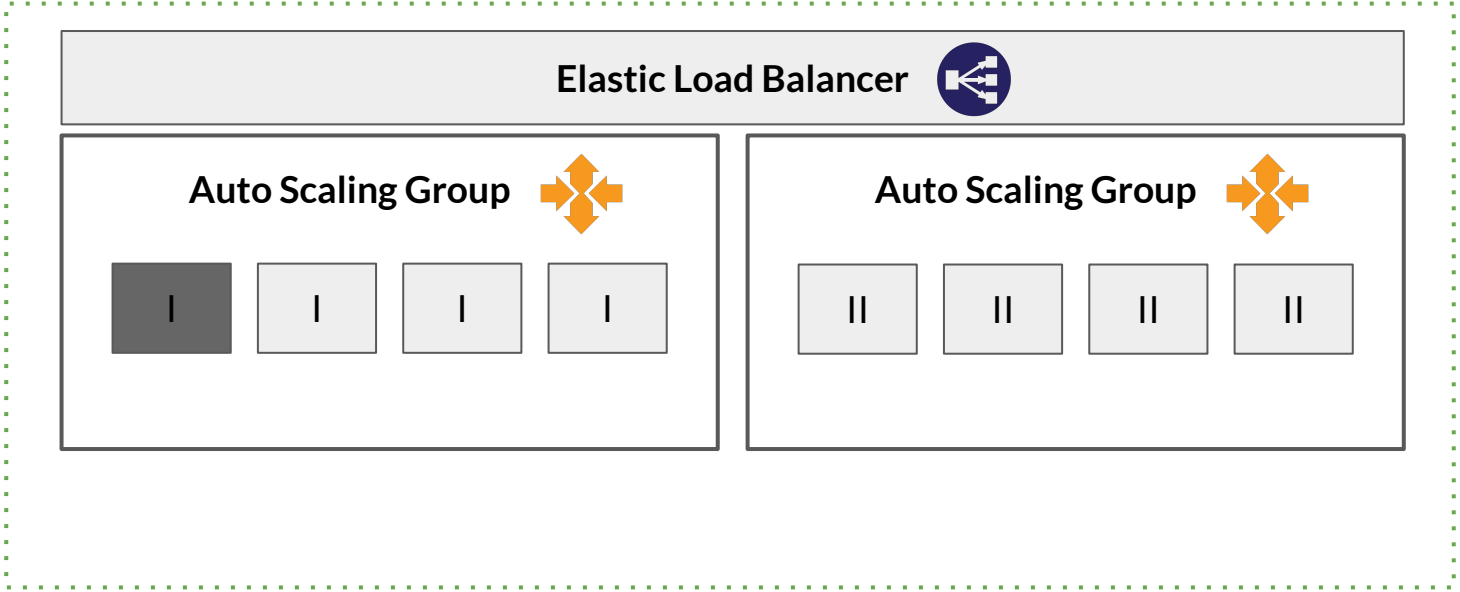
Estrategia Blue Green



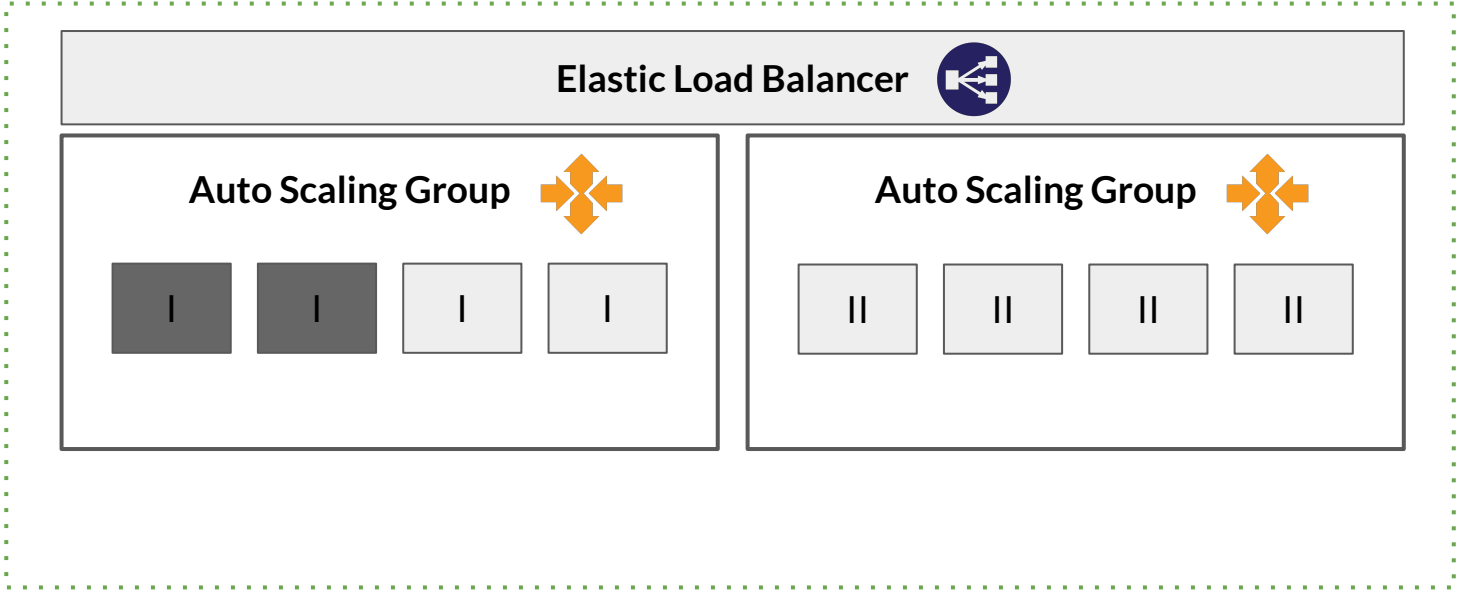
Estrategia Blue Green



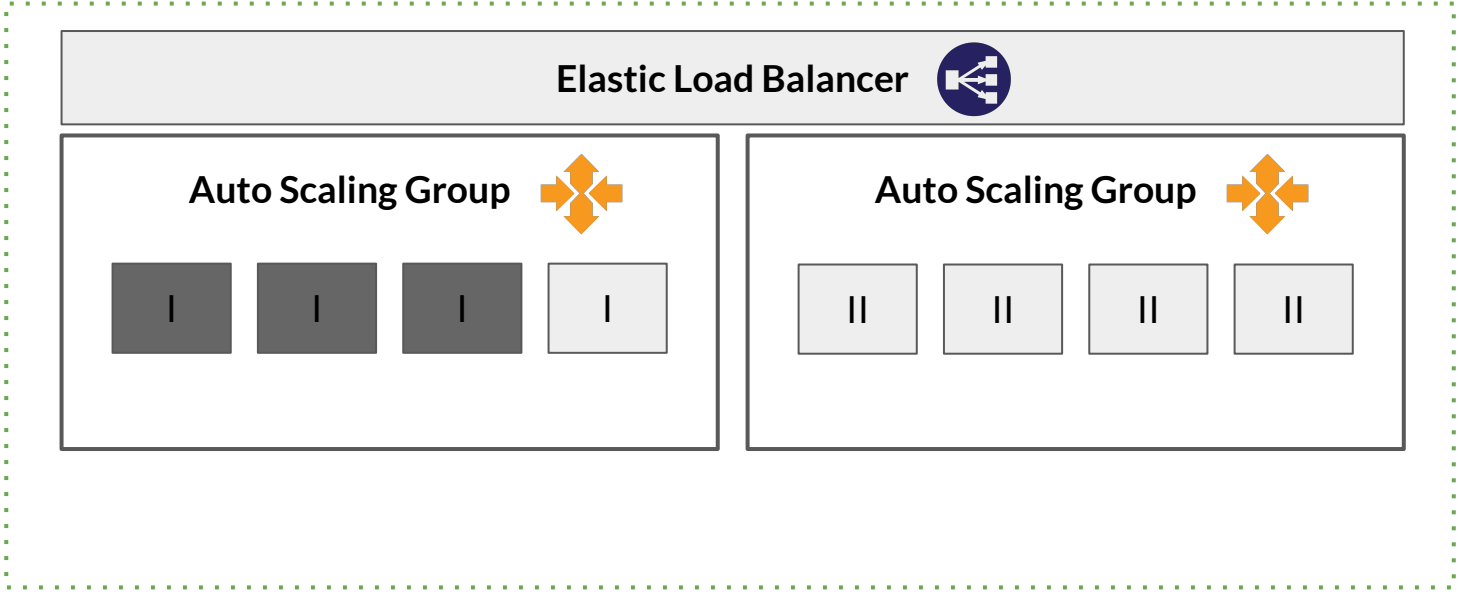
Estrategia Blue Green



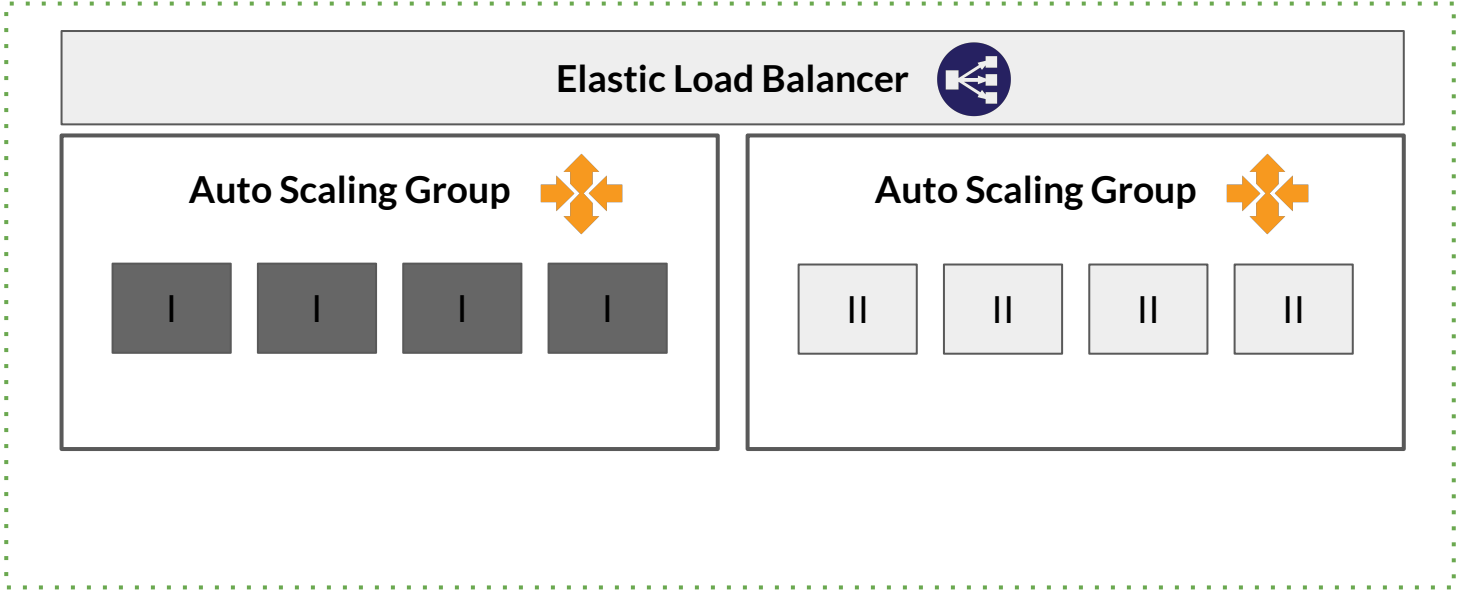
Estrategia Blue Green



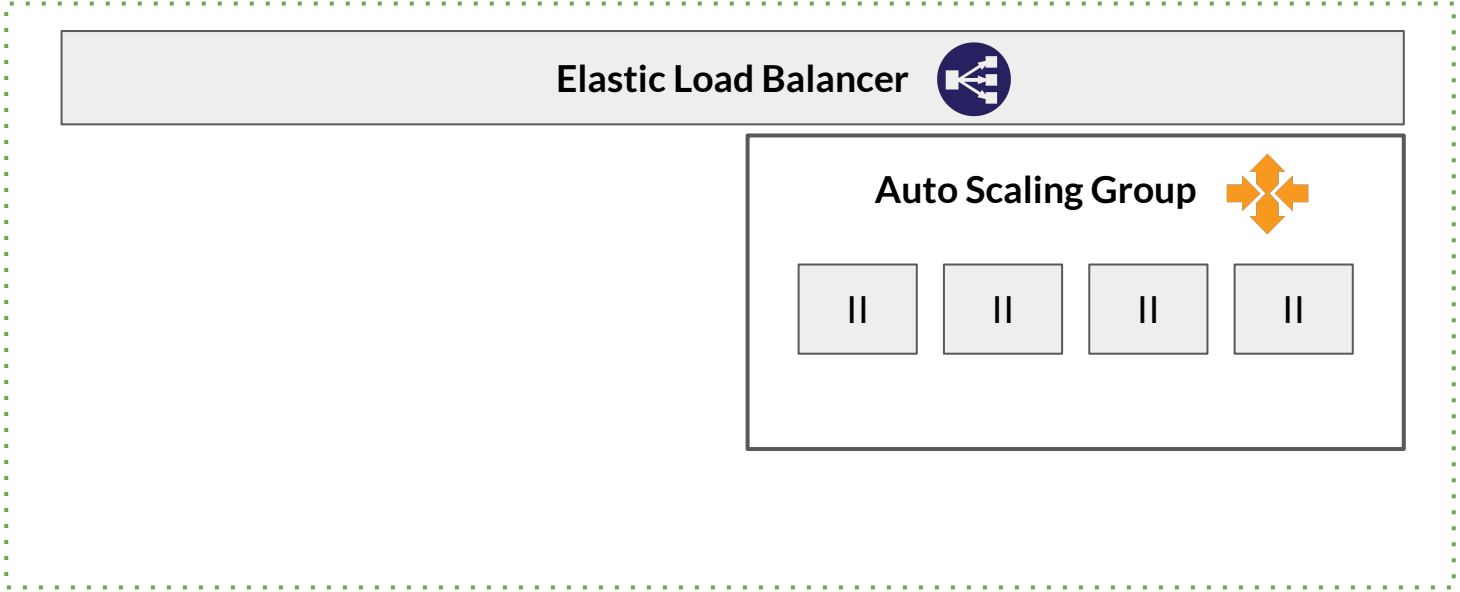
Estrategia Blue Green



Estrategia Blue Green



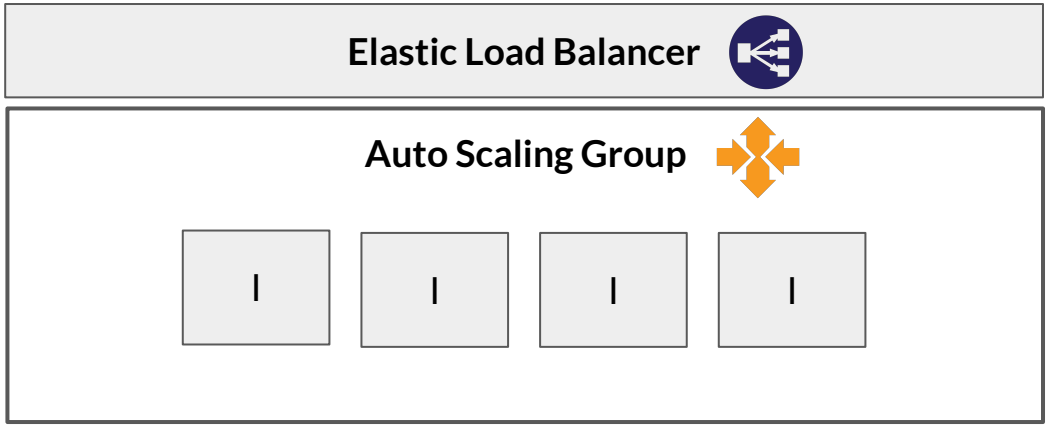
Estrategia Blue Green



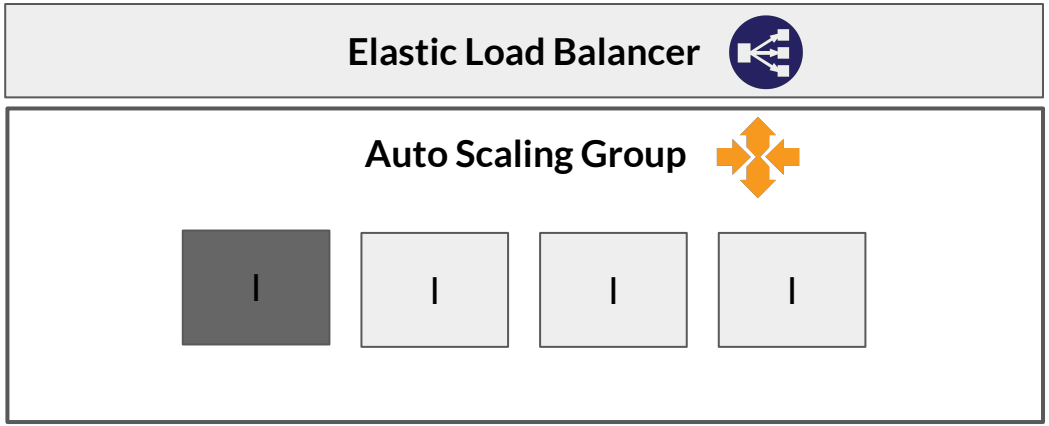
2. Deployar - Estrategias

a. Rolling Update
    i. Reutiliza infraestructura
    ii. Deploy más rápido
    iii. Rollback lento

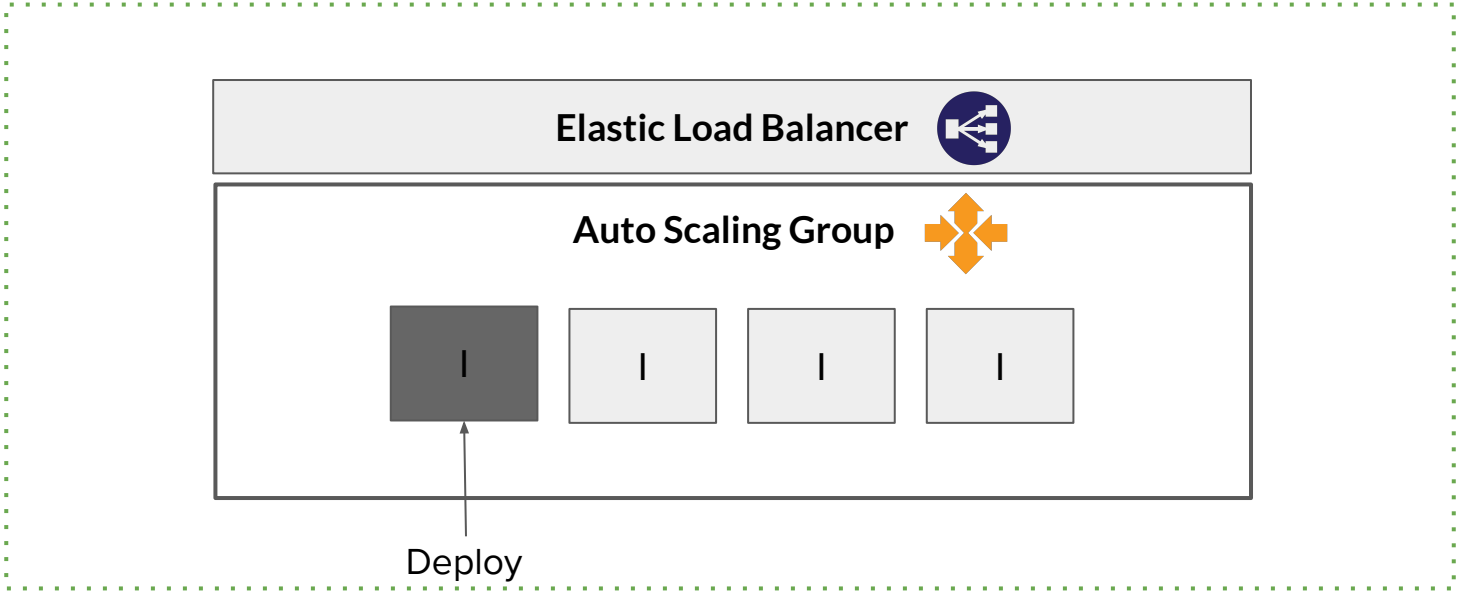
Estrategia Rolling Update



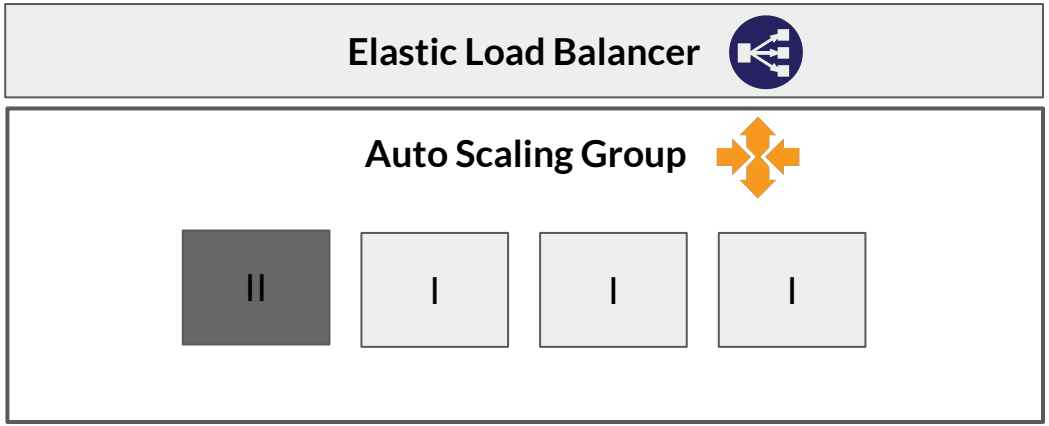
Estrategia Blue Green



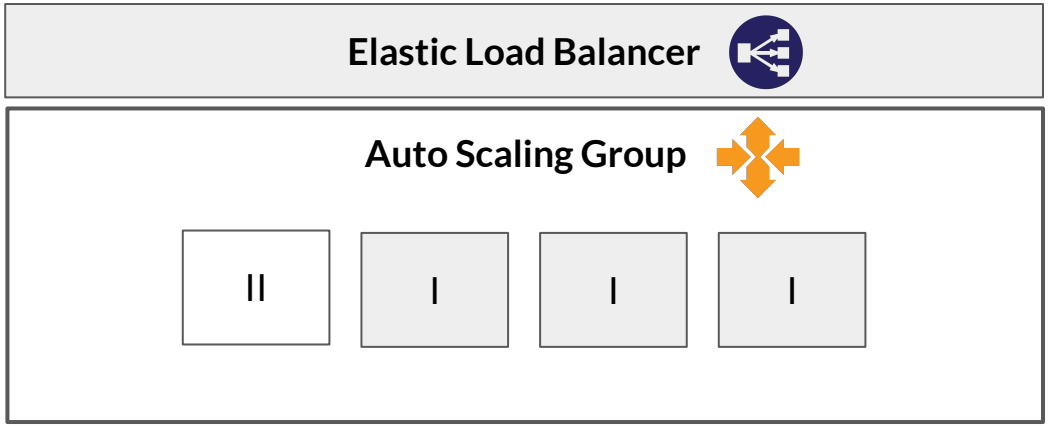
Estrategia Blue Green



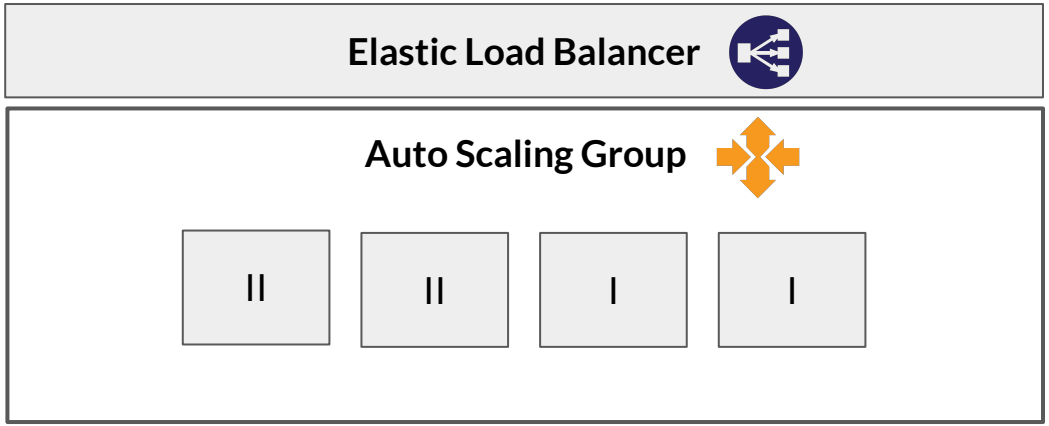
Estrategia Blue Green



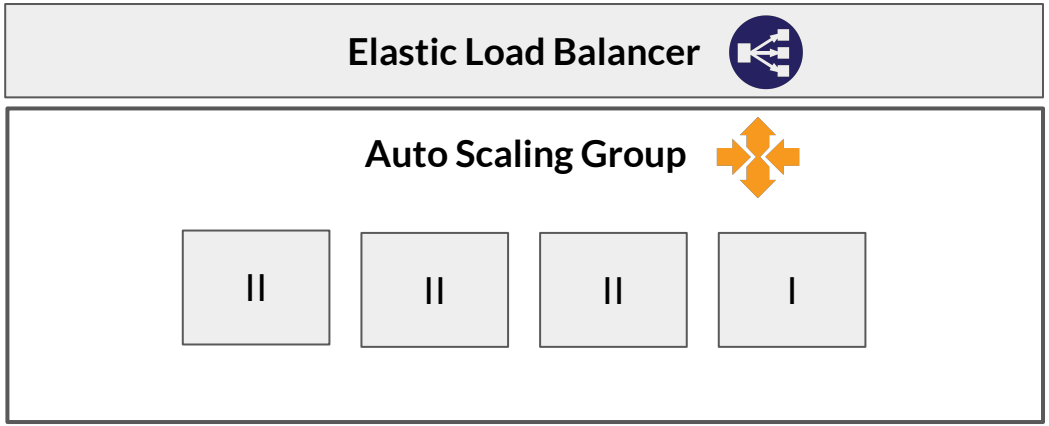
Estrategia Blue Green



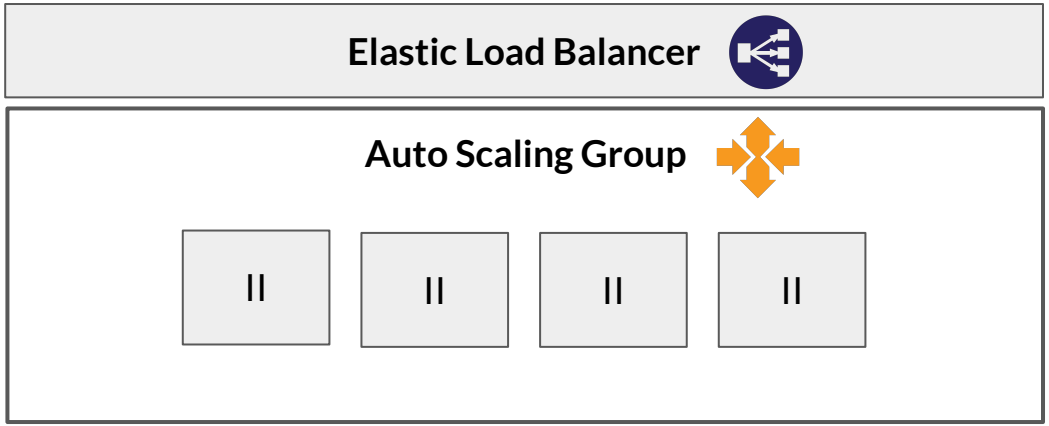
Estrategia Blue Green



Estrategia Blue Green



Estrategia Blue Green



Conclusiones

Gracias!

lucia.brizuela@mercadolibre.com
sebastian.schepens@mercadolibre.com

jobs.mercadolibre.com